Ultra C
Library Reference

Version 2.1

MICROWARE™

Intelligent Products For A Smarter World

Copyright and Publication Information

Copyright ©2000 Microware Systems Corporation. All Rights Reserved. Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from Microware Systems Corporation.

This manual reflects version 2.1 of Ultra C.

Revision: K
Publication date: September 2000

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, Microware will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction Notice

The software described in this document is intended to be used on a single computer system. Microware expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of Microware and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.

For additional copies of this software/documentation, or if you have questions concerning the above notice, please contact your OS-9 supplier.

Trademarks

OS-9, OS-9000, DAVID, and MAUI are registered trademarks of Microware Systems Corporation. SoftStax, FasTrak, UpLink, and Hawk are trademarks of Microware Systems Corporation. All other product names referenced herein are either trademarks or registered trademarks of their respective owners.

Address

Microware Systems Corporation
1500 N.W. 118th Street
Des Moines, Iowa 50325
515-223-8000

Table of Contents

Chapter 1: Overview 7

8 Using Functions and Macros
10 Library Files
12 Program Globals
12 Class 1 Globals
13 Class 2 Globals
14 Header Files
35 The C Library
35 Communicating with Environment Functions
36 Character Classification Functions
37 Character Conversion Functions
38 File Positioning Functions
39 GetStat/SetStat Functions
45 Input/Output Device Functions
45 Input/Output Functions
52 Interprocess Communication Functions
59 Interrupt Manipulation Functions
60 Mathematical Functions
62 Memory Management Functions
65 Miscellaneous Functions
68 Module Manipulation Functions
72 Multibyte Character and String Functions
73 OS System Functions
74 POSIX Messaging Functions
76 Process Manipulation Functions
79 Resource Locks
80 Searching and Sorting Functions
81 String Handling Functions

83 Terminal Manipulation Functions (Termcap)
84 Time Functions
89 Processor-Specific Functions
90 Implementation-Defined Behavior
92 ANSI “Core” Signals
92 OS-9 “Core” Signals
92 OS-9 68K Family Signals
92 OS-9 80x86 Family Signals
93 OS-9 PowerPC™ Family Signals
95 Processor-Specific Family Signals
95 68K Family Processors
96 80x86 Family Processors
96 PowerPC Family Processors
97 ARM Family Processors
97 SH Family Processors
98 SPARC Family Processors
102 Equivalents for the sys_clib.l Functions
110 The Function Description
110 Syntax
111 OS-9 Attributes
111 Library
111 Errors
111 References to Other Calls

Chapter 2: Functions **113**

951 Default Handling
952 Ignore Handling
952 Available Signals
952 ANSI “Core” Signals
953 OS-9 for 68K/OS-9 “Core” Signals
954 680x0: OS-9 for 68K, OS-9
954 680x0: OS-9
955 80x86: OS-9

956 PowerPC: OS-9
956 Function Handling
956 Asynchronous Signals

Appendix A: Prototyping sys_clib.l Functions **1093**

1094 Function Declarations and K&R Compiler vs. Prototypes and ANSI C
1096 "ANSI"fying Code and Function Prototypes

Index **1097**

Product Discrepancy Report **1195**

Chapter 1: Overview

Ultra C library calls perform a variety of functions, from classifying characters to managing memory. This section contains information helpful for using the Ultra C library. This information includes:

- **Using Functions and Macros**
- **Library Files**
- **Program Globals**
- **Header Files**
- **The C Library**

Using Functions and Macros

Ultra C supports both functions and macros:

- A **function** consists of compiled C statements.
- A **macro** is an identifier defined to represent a value or expression.

In most cases, the differences between the two are transparent.

WARNING

Textual replacement in macro expansion can cause arguments of macros to be evaluated repeatedly. Macro implementations of ANSI Standard Library functions are required to evaluate arguments exactly once and to have sufficient parentheses to avoid operator precedence trouble.

You can bypass the macro replacement when you compile. This may be desirable when a large macro is used many times and the space saved by calling the function is outweighed by the extra time needed to call the function. For example, to override the replacement/instantiation of the `putc` macro use:

```
(putc)(c, fp);
```

or

```
#undef putc
```

This results in a call to the `putc` function in the library rather than the replacement of the `putc` macro.

All American National Standards Institute (ANSI) macros have a library counterpart of the same name, with the exception of the following:

- `assert()`
- `setjmp()`
- `va_arg()`
- `va_end()`
- `va_start()`

This is not true for non-ANSI macros.

Library Files

The following are important files used during the compilation process. These files are located in the `MWOS` structure under `LIB` for the hardware platform.

Libraries provided with Ultra C are listed in **Table 1-1**.

Table 1-1 Libraries Provided with this Release

Library	Contains
`clib.l`	The ANSI standard library
`cstart.r`	Start-up code for compiled programs (with the `-bc` option K & R compatible source mode)
`acstart.r`	By default (same as `cstart.r` but compiled with ANSI defined)
`acstarta.r`	icode linked version of the above
`ansi_cstart.r`	Same as `acstart.r`, mainly for the 8.3 filenames on DOS
`ansi_cstarta.r`	Same as `acstarta.r`
`cstarta.r`	With `-bc -j` option (icode linked version, backward compatibility - K&R mode)
`sacstart.r`	ANSI version of `syscstart.r`
`scstart.r`	Used to make system state processes
`scstarta.r`	Version of `scstart.r` used when `-j` option used
`sysansi_cstart.r`	Same as `sacstart.r`
`sysansi_cstarta.r`	Same as `sacstrta.r`
`syscstart.r`	Same as `scstart.r` (8.3 format)

Table 1-1 Libraries Provided with this Release

Library	Contains
`syscstarta.r`	same as `scstarta.r` (8.3 format)
`cpu.l`	CPU dependent functions
`mq.l`	POSIX messaging functions
`os_lib.l`	The system call functions
`sys_clib.l`	The non-ANSI C library functions
`termlib.l`	The termcap database screen manipulation functions
`sclib.l`	To minimize code size, smaller versions of some library functions are available in the libraries `sclib.l` and `sclib.il`.
`unix.l`	The UNIX-like library functions

For More Information

The primary purpose of the `sys_clib.l` library is to provide a library for compatibility with the Microware K&R C compiler. For additional information, refer to Appendix A: Prototyping sys_clib.l Functions.

For More Information

For information on `sclib.l`, small library functions, see Chapter 2 of ***Using Ultra C/C++***.

Program Globals

Ultra C defines a number of globals for use by library functions and/or the application. They are divided into two classes:

1. Globals that are unlikely to change from release to release, and
2. Globals that are used by library functions that may change from release to release (see header file `cglob.h` for current information).

Class 1 Globals

`u_int32 _totmem`	Total number of bytes that are in the data area for the program (includes stack, global data, and parameters)
`u_int16 _pathcnt`	Number of open paths the process inherited from the parent process (usually 3)
`void *_sysglob`	Pointer to the kernel's system globals. This variable is only initialized when the program is a system state process (otherwise it is NULL). **OS-9 for 68K only.**
`mh_exec *_modhead`	Pointer to the primary module for the current process
`char *_modname`	Pointer to the module name for the current process (unmodifiable).
`u_int32 _procid`	Process ID of the current process
`u_int32 _maxstack`	Maximum number of bytes consumed from the stack so far. This variable does not contain a useful value unless stack checking is enabled. Print this variable prior to exiting to examine how much stack space your application required.

`void *_glob_data`	Pointer to the current processes global static storage. `_glob_data` can be used as an argument to `_os_intercept()`.
`char **_environ`	Pointer to the environment variable strings for the current process. This variable is used when forking other processes to pass the current environment along.
`char **environ`	Same as `_environ`, except that it is available only in K & R source mode for OS-9 for 68K. `environ` is provided for compatibility with Microware K & R C-compiler.

Class 2 Globals

`void *_sttop`	Address of the top of the stack
`void *_mtop`	Address of data memory. `_mtop` is used to check for stack overflow.
`void *_stbot`	Deepest stack pointer so far
`void *_csl_a6`	Global static storage pointer for attached `csl`. **OS-9 for 68K only.**
`void *_csl_glob`	Global static storage pointer for attached `csl`. **OS-9 only.**
`u_int32 _sbsize`	Size of process memory area. `_sbsize` is used by `sbrk()`.
`FILE (*_fcbs)[ ]`	Pointer to `_niob` array
`void *_ofcbs`	Pointer to compatibility `_iob` array
`int32 _stklimit`	Amount of verified area left on stack. This value contains the number of bytes that can be consumed from the stack before further checking needs to be done.

Header Files

Most of the Ultra C library calls are declared in header files. A header file declares a set of related functions and includes any necessary types and additional macros needed to use the functions. The contents of these header files are available through the `#include` preprocessing directive. Header files may be included in any order.

Ultra C provides the following header files.

Table 1-2 Ultra C Header Files

File	Description

`alarm.h`	Used by functions that use the OS-9 alarm functions. `alarm.h` contains prototypes for the following functions:
	`alm_atdate()alm_atjul()`
	`alm_cycle()alm_delete()`
	`alm_set()_os9_alarm_atdate()`
	`_os_alarm_atime()_os9_alarm_atjul()`
	`_os_alarm_cycle()_os_alarm_delete()`
	`_os_alarm_reset()_os_alarm_set()`
	`_os9_salarm_atdate()_os_salarm_atime()`
	`_os9_salarm_atjul()_os_salarm_cycle()`
	`_os9_salarm_cycle()_os_salarm_delete()`
	`_os_salarm_reset()_os_salarm_set()`
	`_os9_salarm_set()`

`assert.h` †	Defines the `assert` macro. `assert.h` contains prototypes for the following function: `assert()`

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
`aton.h`	Used for converting alphabetic characters to numeric values. `aton.h` contains prototypes for the following function: `_aton()`
`cache.h`	Defines the CPU internal cache control information. `cache.h` contains prototypes for the following functions: `_os_cache()` `_os_scache()`
`cdi.h` **(OS-9 for 68K only)**	Contains CD-I specific functions. `cdi.h` contains prototypes for the following function: `_os9_gs_cdfd()`
`cglob.h`	Declares Ultra C global variables.

`ctype.h` †	Declares functions that are useful for testing and mapping characters. `ctype.h` contains prototypes for the following functions:
	`isalnum()isalpha()`
	`_isascii(), isascii()iscntrl()`
	`isdigit()_iseuc_kana()`
	`_iseuc1(), _iseuc2()isgraph()`
	`_isjis_kana()islower()`
	`isprint()ispunct()`
	`_issjis1()_issjis2()`
	`isspace()isupper()`
	`isxdigit()toascii(), _toascii()`
	`tolower()_tolower()`
	`toupper()_toupper()`

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description

`dexec.h`	Defines information related to debugger controlled processes. `dexec.h` contains prototypes for the following functions:
	`_os_dexec()_os_dexit()`
	`_os_dfork()_os_dforkm()`
	`_os9_dfork()`

`dir.h`	Defines structures and functions for BSD 4.2 directory calls. `dir.h` contains prototypes for the following functions:
	`closedir()opendir()`
	`readdir()rewinddir()`
	`seekdir()telldir()`

`direct.h` **(OS-9 for 68K only)**	Defines the RBF data structures. This is obsolete; it has been replaced with `rbf.h`.
`errno.h` †	Defines macros relating to error reporting. `errno.h` contains a prototype for the following function: `_os_perr()`

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description

`events.h`	Used for functions using the OS-9 event functions. `events.h` contains prototypes for the following functions:
	`_ev_creat()_ev_delete()`
	`_ev_info()_ev_link()`
	`_ev_pulse()_ev_read()`
	`_ev_set()_ev_setr()`
	`_ev_signal()_ev_unlink()`
	`_ev_wait()_ev_waitr()`
	`_os_ev_allclr()_os_ev_allset()`
	`_os_ev_anyclr()_os_ev_anyset()`
	`_os_ev_change()_os_ev_creat()`
	`_os_ev_delete()_os_ev_info()`
	`_os_ev_link()_os_ev_pulse()`
	`_os_ev_read()_os_ev_set()`
	`_os_ev_setand()_os_ev_setor()`
	`_os_ev_setr()_os_ev_setxor()`
	`_os_ev_signal()_os_ev_tstset()`
	`_os_ev_unlink()_os_ev_wait()`
	`_os9_ev_wait()_os_ev_waitr()`
	`_os9_ev_waitr()_os9_ev_info()`

`float.h` †	Defines the limits for floating point numbers.
`funcs.h`	Symbolic names for the system calls. `funcs.h` contains a prototype for the following function: `_os9_setsvc()`

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
`getset.h`	Used to read and set values of special purpose registers, device control registers, processor registers, and time-based registers. `getset.h` contains prototypes for the following functions: `_get_<name>()_set_<name>()`

`io.h`	Used for functions performing input/output. `io.h` contains prototypes for the following functions:
	`_os_alias()_os_attach()`
	`_os_detach()_os_get_ioproc()`
	`_os_getdl()_os_getpd()`
	`_os_rdalst()_os_tranpn()`

`ioctl.h`	Contains definitions for `ioctl()` library function.
`ioedt.h`	Defines the current I/O interface editions. `ioedt.h` contains a prototype for the following function: `_os_gs_edt()`
`limits.h` †	Defines macros that expand to various limits and parameters.
`locale.h` †	Used for localization. `locale.h` contains prototypes for the following functions: `localeconv()setlocale()`

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description

`lock.h`	Used for the OS-9 resource lock functions. `lock.h` contains prototypes for the following functions:
	`_os_acqlk()_os_caqlk()`
	`_os_crlk()_os_dellk()`
	`_os_rellk()_os_waitlk()`

`math.h` †	Defines the mathematical functions and macros available with Ultra C. `math.h` contains prototypes for the following functions:
	`acos()` `asin()`
	`atan()` `atan2()`
	`ceil()` `cos()`
	`cosh()` `exp()`
	`fabs()` `floor()`
	`fmod()` `frexp()`
	`hypot()` `ldexp()`
	`log()` `log10()`
	`modf()` `pow()`
	`sin()` `sinh()`
	`sqrt()` `tan()`
	`tanh()`

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description

`memory.h`	Functions related to acquiring, examining, and releasing memory. `memory.h` contains prototypes for the following functions:
	`_os9_allbit()_os9_delbit()`
	`_os_get_blkmap()_os_mem()`
	`_os_move()_os9_schbit()`
	`_os_srqmem()_os9_srqmem()`
	`_os_srtmem()_os9_translate()`
	`_srqmem()_srtmem()`
	`srqcmem()`

`moddir.h`	Module directory information. `moddir.h` contains prototypes for the following functions:
	`_os_altmdir()_os_chmdir()`
	`_os_cmdperm()_os_delmdir()`
	`_os_fmod()_os_get_mdp()`
	`_os_makmdir()`

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description

`modes.h`	Defines the access modes and permissions for files and directories. `modes.h` contains prototypes for the following functions:
	`access()attach()`
	`chdir() chown()`
	`chmod()chxdir()`
	`close() creat()`
	`create()detach()`
	`dup()  lseek()`
	`makdir()mknod()`
	`open()_os_chdir()`
	`_os_close()_os_create()`
	`_os9_create()_os_delete()`
	`_os_dup()_os_ioconfig()`
	`_os_makdir()_os_open()`
	`_os_read()_os_readln()`
	`_os_seek()_os9_ss_close()`
	`_os9_ss_open()_os_write()`
	`_os_writeln()read()`
	`readln()unlink()`
	`unlinkx()write()`
	`writeln()`

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
module.h	Module-related data structures and functions. module.h contains prototypes for the following functions: crc()_get_module_dir() make_module()modcload() _mkdata_module()modlink() modload()modloadp() munlink()munload() _os_crc()_os_datmod() _os_get_moddir()_os_initdata() _os_iodel()_os_link() _os_linkm()_os_load() _os_loadp()_os_mkmodule() _os_modaddr()_os_setcrc() _os_slink()_os_slinkm() _os_tlinkm()_os_tlink() _os_unlink()_os_unload() _os_vmodul()_os9_vmodul() _prgname()_setcrc() _sliblink()_subcall()
mqueue.h	POSIX messaging data structures and functions. mqueue.h contains prototypes for the following functions: mq_close; mq_getattr; mq_notify; _mq_notify_write; mq_open; mq_receive; mq_send; mq_setattr; mq_unlink

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
os9def.h	A series of macros mapping UNIX macros and function calls to their OS-9 versions. alarm()alloca() bcmp() bcopy() buildpath()bzero() cbrt() crtolf() dup2() execl() execle()execv() execve()execvp() ffs() getgid() getopt()getppid() getwd() ioctl() isatty()lftocr() pclose() pipe() popen()putenv() readv()setgid() tempnam()writev()
os9time.h	Defines interval timer support for OS-9. gettimeofday()
os9types.h	Synchronous I/O multiplexing. os9types.h contains a prototype for the following function: select()

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
process.h	Process descriptor and functions that manipulate processes. process.h contains prototypes for the following functions: chainc(), chain() _cpymem()getpid() _get_process_desc() _get_process_table() getuid()_os9_allpd() _os_alltsk()_os_alocproc() _os_aproc()_os9_aproc() _os_chain()_os_chainm() _os_chkmem()_os_cpy_ioproc() _os_cpymem()_os_ddlk() _os_deltsk()_os_exec() os9exec()_os_exit() _os_findpd()_os9_findpd() _os_fork()os9fork(), os9forkc() _os_forkm()_os_get_prtbl() _os_gprdsc()_os9_gspump() _os_id()_os9_id() _os_ioexit()_os_iofork() _os9_ioqueue()_os_nproc() _os9_panic()_os_permit() _os_protect()_os9_retpd() _os_rtnprc()_os_setpr() _os_setuid()_os_suspend() _os_sysdbg()_os_uacct() _os_wait()setpr() setuid()_sysdbg() wait()

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
`procid.h`	Process descriptor data structure. `procid.h` is obsolete; it has been replaced with `process.h`.

`pwd.h`	Contains the password file definitions.
	`endpwent()fgetpwent()`
	`getnextpwnam()getpw()`
	`getpwent()getpwnam()`
	`getpwuid()setpwent()`

`rbf.h`	Used for functions using the random block file manager (RBF). `rbf.h` contains prototypes for the following functions:
	`_gs_gfd()_os_gs_cstats()`
	`_os_gs_dsize()_os_gs_fd()`
	`_os_gs_fdaddr()_os_gs_fdinf()`
	`_os_gs_parity()_os_ss_cache()`
	`_os_ss_fd()_os_ss_flushmap()`
	`_os_ss_hdlink()_os_ss_lock()`
	`_os_ss_ticks()_os_ss_wtrack()`
	`_ss_lock()_ss_pfd()`
	`_ss_tiks()_ss_wtrk()`

`regexp.h`	Contains definitions for regular expression support.
	`regcomp()regerror()`
	`regex(), regexec()`

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description

`regs.h`	Used to include the proper register header file for the target processor. `regs.h` contains prototypes for the following functions:
	`change_const()change_static()`
	`get_const()get_static()`
	`inc()  inl()`
	`inw()irq_change()`
	`irq_disable()irq_enable()`
	`irq_maskget()irq_restore()`
	`irq_save()make_gdesc()`
	`_os_cache()`(OS-9 only)
	`_os_firq()_os_irq()`
	`_os9_irq()`(OS-9 for 68K only)
	`_os_scache()_os_sysid()`
	`outc()  outl()`
	`outw()`

`sbf.h`	Used for functions using the sequential block file manager (SBF). `sbf.h` contains prototypes for the following functions:
	`_os_ss_erase()_os_ss_rfm()`
	`_os_ss_reten()_os_ss_skipend()`
	`_os_ss_skip()_os_ss_wfm()`

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description

`scf.h`	Used for functions using the sequential character file manager (SCF). `scf.h` contains prototypes for the following functions:
	`_os_ss_dcoff()_os_ss_dcon()`
	`_os_ss_dopt()_os_ss_dsrts()`
	`_os_ss_enrts()_os_ss_fillbuff()`
	`_ss_dcoff()_ss_dcon()`
	`_ss_dsrts()_ss_enrts()`

`semaphore.h`	Used for functions using binary semaphores. `semaphore.h` contains prototypes for the following functions:
	`_os_sema_init()_os_sema_p()`
	`_os_sema_term()_os_sema_v()`

`setjmp.h` †	Used for bypassing the normal function call and return mechanism. `setjmp.h` contains prototypes for the following functions: `longjmp()setjmp()`
`setsys.h`	System global symbolic names. This is obsolete. It has been replaced with `sysglob.h`. `setsys.h` contains prototypes for the following functions: `_getsys()_setsys()`
`settrap.h`	Data structures and functions used to trap user-state exceptions. `settrap.h` contains prototypes for the following functions: `_os_strap()_os9_strap()`

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
sg_codes.h	Symbolic names for SetStats and GetStats. sg_codes.h contains prototypes for the following functions: getstat()_gs_devn() _gs_eof()_gs_opt() _gs_pos()_gs_rdy() _gs_size()_os_getstat() _os_gs_cpypd()_os_gs_devnm() _os_gs_devtyp()_os_gs_dopt() _os_gs_eof()_os9_gs_free() _os_gs_luopt()_os_gs_popt() _os_gs_pos()_os_gs_ready() _os_gs_size()_os_setstat() _os_sgetstat()_os_sgs_devnm() _os_ss_attr()_os_ss_break() _os_ss_dopt()_os_ss_link_adj() _os_ss_luopt()_os_ss_popt() _os_ss_relea()_os_ss_rename() _os_ss_reset()_os_ss_sendsig() _os_ss_size()setstat() _ss_attr()_ss_opt() _ss_rel()_ss_rest() _ss_size()_ss_ssig()
sgstat.h	Path descriptor option section definitions.

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description

`signal.h` †	Used for handling various signals. `signal.h` contains prototypes for the following functions:
	`intercept()kill()`
	`_os_clrsigs()_os_intercept()`
	`_os_rte()_os_send()`
	`_os_siglngj()_os_sigmask()`
	`_os_sigreset()_os_sigrs()`
	`_os_sleep()_os9_sleep()`
	`pause() raise()`
	`sigmask()signal()`
	`sleep()tsleep()`

`stat.h`	Contains definitions for `stat()` and `fstat()` support.

`stdarg.h` †	Used for advancing through a list of arguments of variable length and type. `stdarg.h` contains the macros for:
	`va_arg()va_end()`
	`va_start()`

`stddef.h` †	Used for common definitions (`ptrdiff_t`, `size_t`, and `wchar_t`) and the following macro: `offsetof()`

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description

`stdio.h` †	Used for performing input and output. `stdio.h` contains prototypes for the following functions:
	`_cleareof(), cleareof()`
	`clearerr()_errmsg()`
	`fclose()fdopen()`
	`feof() ferror()`
	`fflush()fgetc()`
	`fgetpos()fgets()`
	`_fileno(), fileno()`
	`fopen()`
	`fprintf()fputc()`
	`fputs()fread()`
	`freopen()fscanf()`
	`fseek()fsetpos()`
	`ftell()fwrite()`
	`getc(), getchar()`
	`gets()  getw()`
	`mktemp()perror()`
	`prerr()printf()`
	`putc(), putchar()`
	`puts()  putw()`
	`remove()rename()`
	`rewind()scanf()`
	`setbuf()setvbuf()`
	`sprintf()sscanf()`
	`tmpfile()tmpnam()`
	`ungetc()vfprintf()`
	`vprintf()vsprintf()`

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description

`stdlib.h` †	Used for the standard library functions. `stdlib.h` contains prototypes for the following functions:
	`abort()  abs()`
	`atexit() atof()`
	`atoi()  atol()`
	`_atou()bsearch()`
	`calloc()  div()`
	`_dtoa()  ebrk()`
	`exit()  _exit()`
	`free()`
	`_freemem(), freemem()`
	`_freemin()ibrk()`
	`getenv() labs()`
	`_lcalloc()ldiv()`
	`_lfree()_lmalloc()`
	`_lrealloc()malloc()`
	`_mallocmin()mblen()`
	`mbstowcs()mbtowc()`
	`qsort()  rand()`
	`realloc()sbrk()`
	`srand()`
	`stacksiz(), _stacksiz()`
	`strtod()strtol()`
	`strtoul()system()`
	`wcstombs()wctomb()`

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description

`string.h` †	Used for manipulating arrays of character type. `string.h` contains prototypes for the following functions:
	`memchr()memcmp()`
	`memcpy()memmove()`
	`memset()strcat()`
	`strchr()strcmp()`
	`strcoll()strcpy()`
	`strcspn()strerror()`
	`strlen()strncat()`
	`strncmp()strncpy()`
	`strpbrk()strrchr()`
	`strspn()strstr()`
	`strtok()strxfrm()`

`strings.h`	Used for non-ANSI string functions. `strings.h` contains prototypes for the following functions:
	`_cmpnam()findnstr()`
	`findstr()index()`
	`_os_cmpnam()_os_prsnam()`
	`pause()_prsnam()`
	`rindex()_strass()`
	`strhcpy()`

`svctbl.h`	Contains information for the service table. `svctbl.h` contains a prototypes for the following function: `_os_setsvc()`

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description

`sysglob.h`	Defines the system global variables. `sysglob.h` contains prototypes for the following functions:
	`_getsys()_os_config()`
	`_os_getsys()_os_setsys()`
	`_setsys()`

`termcap.h`	Used for the terminal capability functions. `termcap.h` contains prototypes for the following functions:
	`tgetent()tgetflag()`
	`tgetnum()tgetstr()`
	`tgoto() tputs()`

`termio.h`	Contains definition to support `ioctl`.

`time.h` †	Used for manipulating time. `time.h` contains prototypes for the following functions:
	`asctime()clock()`
	`ctime()difftime()`
	`getime()`
	`gmtime()_gregorian()`
	`_julian()localtime()`
	`mktime()_os_getime()`
	`_os9_getime()_os_gregorian()`
	`_os_julian()_os_setime()`
	`_os9_setime()`
	`setime()strftime()`
	`_sysdate()sys_mktime()`
	`time()`

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
`times.h`	Contains definitions to support times. `times()`
`types.h`	Defines `typedefs` for consistently sized and signed types.
`un.h`	Contains the `sockaddr_un` structure definition.
`virtual.h`	Contains prototype for the following functions: `_os_transadd()`

† Indicates a standard ANSI header.

The C Library

This section includes a listing of the available C functions, grouped according to functionality. Each listing includes the function's name, description, and the header file in which it is defined.

Communicating with Environment Functions

The Ultra C libraries contain the functions listed in **Table 1-3** for communicating with the environment.

Table 1-3 Environment Communications Functions

Function	ANSI Standard	Description	Header File
`abort()`	•	Abnormal program termination	`stdlib.h`
`atexit()`	•	Register function to call at normal program termination	`stdlib.h`
`_exit()`		Task termination	`stdlib.h`
`exit()`	•	Task termination	`stdlib.h`
`getenv()`	•	Get value for environment name	`stdlib.h`
`_os_exit()`		Terminate calling process	`process.h`
`system()`	•	Shell command execution	`stdlib.h`

Character Classification Functions

The character classification functions are really macros defined in <ctype.h>. Be careful when using these macros to avoid macro expansion side-effects. The macros provide a platform independent method of character classification. **Table 1-4** identifies character classification functions.

Table 1-4 Character Classification Functions

Function	ANSI Standard	Description	Header File
isalnum()	•	Test if parameter is alphanumeric	ctype.h
isalpha()	•	Test if parameter is alphabetic	ctype.h
_isascii(), isascii()		Test if parameter is ASCII	ctype.h
iscntrl()	•	Test if parameter is control code	ctype.h
isdigit()	•	Test if parameter is digit	ctype.h
isgraph()	•	Test if parameter is printing character	ctype.h
_isjis_kana()		Test if parameter is Hankaku-Kana	ctype.h
islower()	•	Test if parameter is lowercase	ctype.h
isprint()	•	Test if parameter is printable character	ctype.h
ispunct()	•	Test if parameter is punctuation character	ctype.h

Table 1-4 Character Classification Functions (continued)

Function	ANSI Standard	Description	Header File
`_issjis1()`		Test if parameter is the first byte of Kanji	`ctype.h`
`_issjis2()`		Test if parameter is the second byte of Kanji	`ctype.h`
`isspace()`	•	Test if parameter is white space	`ctype.h`
`isupper()`	•	Test if parameter is uppercase	`ctype.h`
`isxdigit()`	•	Test if parameter is hexadecimal	`ctype.h`

Character Conversion Functions

The character conversion functions allow you to convert a string of characters to their numeric representation and to change the case of characters. **Table 1-5** identifies character conversion functions.

Table 1-5 Character Conversion Functions

Function	ANSI Standard	Description	Header File
`atof()`	•	Alpha to floating conversion	`stdlib.h`
`atoi()`	•	Alpha to integer conversion	`stdlib.h`
`atol()`	•	Alpha to long conversion	`stdlib.h`
`_aton()`		Alpha to numeric translation	`aton.h`

Table 1-5 Character Conversion Functions (continued)

Function	ANSI Standard	Description	Header File
`_atou()`		Alpha to unsigned conversion	`stdlib.h`
`_dtoa()`		Double to ASCII conversion	`stdlib.h`
`toascii()`, `_toascii()`		Integer to ASCII translation	`ctype.h`
`_tolower()`		Convert character to lowercase	`ctype.h`
`tolower()`	•	Convert character to lowercase	`ctype.h`
`_toupper()`		Convert character to uppercase	`ctype.h`
`toupper()`	•	Convert character to uppercase	`ctype.h`

File Positioning Functions

The functions in **Table 1-6** are provided for file positioning.

Table 1-6 File Positioning Functions

Function	ANSI Standard	Description	Header File
`fgetpos()`	•	Get current position in file	`stdio.h`
`fseek()`	•	Reposition file pointer	`stdio.h`
`fsetpos()`	•	Reposition file pointer	`stdio.h`
`ftell()`	•	Get current position in file	`stdio.h`

Table 1-6 File Positioning Functions (continued)

Function	ANSI Standard	Description	Header File
_os_seek()		Reposition file pointer	modes.h
rewind()	•	Return file pointer to zero	stdio.h

GetStat/SetStat Functions

The standard library provides the GetStat and SetStat functions for determining and setting individual device parameters that are not uniform on all devices or that are highly hardware dependent.

Table 1-7 GetStat/SetStat Functions

Function	ANSI Standard	Description	Header File
getstat()		Get file status	sg_codes.h
_gs_devn()		Get device name	sg_codes.h
_gs_eof()		Check for end of file	sg_codes.h
_gs_gfd()		Get file descriptor sector	none
_gs_opt()		Get path options	sg_codes.h
_gs_pos()		Get current file position	sg_codes.h
_gs_rdy()		Test for data available	sg_codes.h
_gs_size()		Get current file size	sg_codes.h

Table 1-7 GetStat/SetStat Functions (continued)

Function	ANSI Standard	Description	Header File
_os_getstat()		Get file/device status	sg_codes.h
_os_getsys()		Examine system global variable	sysglob.h
_os9_gs_cdfd()		Return file descriptor	cdi.h
_os_gs_cpypd()		Copy contents of path descriptor	sg_codes.h
_os_gs_cstats()		Get cache status information	rbf.h
_os_gs_devnm()		Return device name	sg_codes.h
_os_gs_devtyp()		Return device type	sg_codes.h
_os_gs_dopt()		Read device path options	sg_codes.h
_os_gs_dsize()		Get size of SCSI devices	rbf.h
_os_gs_edt()		Get I/O interface edition number	ioedt.h
_os_gs_eof()		Test for end of file	sg_codes.h
_os_gs_fd()		Read file descriptor sector	rbf.h
_os_gs_fdaddr()		Get file descriptor block address for open file	rbf.h
_os_gs_fdinf()		Get specified file descriptor sector	rbf.h

Table 1-7 GetStat/SetStat Functions (continued)

Function	ANSI Standard	Description	Header File
_os9_gs_free()		Return amount of free space on device	sg_codes.h
_os_gs_luopt()		Read logical unit options	sg_codes.h
_os_gs_parity()		Calculate parity of file descriptor	rbf.h
_os_gs_popt()		Read path descriptor option section	sg_codes.h
_os_gs_pos()		Get current file position	sg_codes.h
_os_gs_ready()		Test for data ready	sg_codes.h
_os_gs_size()		Return current file size	sg_codes.h
_os_setstat()		Set file/device status	sg_codes.h
_os_setsvc()		Adds or replaces a system service	funcs.h
_os_setsys()		Set system global variables	sysglob.h
_os_sgetstat()		GetStat call using system path number	sg_codes.h
_os_sgs_devnm()		Return device name	sg_codes.h
_os_ss_attr()		Set file attributes	sg_codes.h
_os_ss_break()		Break serial connection	sg_codes.h

Table 1-7 GetStat/SetStat Functions (continued)

Function	ANSI Standard	Description	Header File
_os_ss_cache()		Enable/disable RBF caching	rbf.h
_os9_ss_close()		Notify driver that path has been closed	modes.h
_os_ss_dcoff()		Send signal when data carrier detect line goes false	scf.h
_os_ss_dcon()		Send signal when data carrier detect line goes true	scf.h
_os_ss_dopt()		Set device path options	sg_codes.h
_os_ss_dsrts()		Disable RTS line	scf.h
_os_ss_enrts()		Enable RTS line	scf.h
_os_ss_erase()		Erase tape	sbf.h
_os_ss_fd()		Write file descriptor sector	rbf.h
_os_ss_fillbuff()		Fill path buffer with data	scf.h
_os_ss_flushmap()		Flush cached bit map information for RBF device	rbf.h
_os_ss_hdlink()		Make hard link to existing file	rbf.h
_os_ss_lock()		Lock out record	rbf.h

Table 1-7 GetStat/SetStat Functions (continued)

Function	ANSI Standard	Description	Header File
_os_ss_luopt()		Write logical unit options	sg_codes.h
_os9_ss_open()		Notify driver that path has been opened	modes.h
_os_ss_popt()		Write option section of path descriptor	sg_codes.h
_os_ss_relea()		Release device	sg_codes.h
_os_ss_rename()		Rename file	sg_codes.h
_os_ss_reset()		Restore head to track zero	sg_codes.h
_os_ss_reten()		Retension pass on tape drive	sbf.h
_os_ss_rfm()		Skip tape marks	sbf.h
_os_ss_sendsig()		Send signal on data ready	sg_codes.h
_os_ss_size()		Set file size	sg_codes.h
_os_ss_skip()		Skip blocks	sbf.h
_os_ss_skipend()		Skip to end of tape	sbf.h
_os_ss_ticks()		Wait specified number of ticks for record release	rbf.h
_os_ss_wfm()		Write tape marks	sbf.h
_os_ss_wtrack()		Write (format) track	rbf.h

Table 1-7 GetStat/SetStat Functions (continued)

Function	ANSI Standard	Description	Header File
setstat()		Set file status	sg_codes.h
_ss_attr()		Set file attributes	sg_codes.h
_ss_dcoff()		Send data carrier lost signal to process	scf.h
_ss_dcon()		Send data carrier present signal to process	scf.h
_ss_dsrts()		Disable RTS line	scf.h
_ss_enrts()		Enable RTS line	scf.h
_ss_lock()		Lock out record	rbf.h
_ss_opt()		Set path options	sg_codes.h
_ss_pfd()		Put file descriptor sector	rbf.h
_ss_rel()		Release device	sg_codes.h
_ss_rest()		Restore device	sg_codes.h
_ss_size()		Set current file size	sg_codes.h
_ss_ssig()		Send signal on data ready	sg_codes.h
_ss_tiks()		Wait for record release	rbf.h
_ss_wtrk()		Write track	rbf.h

Input/Output Device Functions

The following functions are available for performing input/output on devices.

Table 1-8 Input/Output Functions

Function	ANSI Standard	Description	Header File
`inc()`		Get character from I/O address	`regs.h`
`inl()`		Get integer from I/O address	`regs.h`
`inw()`		Get word from I/O address	`regs.h`
`outc()`		Write character to I/O address	`regs.h`
`outl()`		Write integer to I/O address	`regs.h`
`outw()`		Write word to I/O address	`regs.h`

Input/Output Functions

The following functions make up the I/O functions of the Ultra C libraries. ANSI prototypes for non-ANSI functions are achieved by defining the macro name `_OPT_PROTOS` during the compile.

Table 1-9 Ultra C I/O Functions

Function	ANSI Standard	Description	Header File
`access()`		Determine file accessibility	`modes.h`
`attach()`		Attach to device	`modes.h`

Table 1-9 Ultra C I/O Functions (continued)

Function	ANSI Standard	Description	Header File
chdir()		Change current data directory	modes.h
chmod()		Change file access permissions	modes.h
chown()		Change owner of file	modes.h
chxdir()		Change current execution directory	modes.h
_cleareof(), cleareof()		Clear end of file condition	stdio.h
clearerr()	•	Clear error condition	stdio.h
close()		Close path	modes.h
closedir()		Close named directory stream	dir.h
creat()		Create file	modes.h
create()		Create file	modes.h
detach()		Detach device	modes.h
dup()		Duplicate path	modes.h
fclose()	•	Close file	stdio.h
fdopen()		Attach path to file pointer	stdio.h

Table 1-9 Ultra C I/O Functions (continued)

Function	ANSI Standard	Description	Header File
feof()	•	Check buffered file for end of file	stdio.h
ferror()	•	Check buffered file for error condition	stdio.h
fflush()	•	Flush file's buffer	stdio.h
fgetc()	•	Get character from file	stdio.h
fgets()	•	Get string from file	stdio.h
_fileno(), fileno()		Determine path number from file	stdio.h
fopen()	•	Open file	stdio.h
fprintf()	•	Formatted output	stdio.h
fputc()	•	Output character to file	stdio.h
fputs()	•	Output string to file	stdio.h
fread()	•	Read data from file	stdio.h
freopen()	•	Re-open file	stdio.h
fscanf()	•	Input string conversion	stdio.h
fwrite()	•	Write data to file	stdio.h
getc(), getchar()	•	Get next character from file/standard input	stdio.h

Table 1-9 Ultra C I/O Functions (continued)

Function	ANSI Standard	Description	Header File
gets()	•	Get string from file	stdio.h
getw()		Read word from file	stdio.h
lseek()		Position file pointer	modes.h
makdir()		Create directory	modes.h
mknod()		Create directory	modes.h
mktemp()		Create unique file name	stdio.h
open()		Open file	modes.h
opendir()		Open directory	dir.h
_os_alias()		Create device alias	io.h
_os_attach()		Attach new device to system	io.h
_os_chdir()		Change working directory	modes.h
_os_close()		Close path to file/device.	modes.h
_os_cpy_ioproc()		Get pointer to I/O process descriptor.	process.h
_os_create()		Create path to new file	modes.h
_os9_create()		Create path to new file	modes.h
_os_delete()		Delete file	modes.h

Table 1-9 Ultra C I/O Functions (continued)

Function	ANSI Standard	Description	Header File
`_os_detach()`		Remove device from system	`io.h`
`_os_dup()`		Duplicate path	`modes.h`
`_os_get_ioproc()`		Get pointer to I/O process descriptor	`io.h`
`_os_getdl()`		Get system I/O device list head pointer	`io.h`
`_os_ioconfig()`		Configure an element of process/system I/O	`modes.h`
`_os_iodel()`		Check for use of I/O module	`module.h`
`_os_ioexit()`		Terminate I/O for exiting process	`process.h`
`_os_iofork()`		Set up I/O for new process	`process.h`
`_os9_ioqueue()`		Enter I/O queue	`process.h`
`_os_makdir()`		Make new directory	`modes.h`
`_os_open()`		Open path to file or device	`modes.h`
`_os_rdalst()`		Copy system alias list	`io.h`
`_os_read()`		Read data from file or device	`modes.h`
`_os_readln()`		Read text line with editing	`modes.h`
`_os_tranpn()`		Translate user path to system path	`io.h`

Table 1-9 Ultra C I/O Functions (continued)

Function	ANSI Standard	Description	Header File
`_os_write()`		Write data to file or device	`modes.h`
`_os_writeln()`		Write line of text with editing	`modes.h`
`pffinit()`		Initialize for float output (obsolete)	`none`
`pflinit()`		Initialize for longs output (obsolete)	`none`
`printf()`	•	Formatted output	`stdio.h`
`putc()`, `putchar()`	•	Put next character to file/standard out	`stdio.h`
`puts()`	•	Output string to file	`stdio.h`
`putw()`		Output word to file	`stdio.h`
`read()`		Read bytes from path	`modes.h`
`readdir()`		Return pointer	`dir.h`
`readln()`		Read bytes from path	`modes.h`
`remove()`	•	Remove file	`stdio.h`
`rename()`	•	Rename file	`stdio.h`
`seekdir()`		Set position of next readdir	`dir.h`
`scanf()`	•	Input strings conversion	`stdio.h`
`setbuf()`	•	Fix file buffer	`stdio.h`

Table 1-9 Ultra C I/O Functions (continued)

Function	ANSI Standard	Description	Header File
setvbuf()	•	Set up buffer for I/O stream	stdio.h
sprintf()	•	Formatted strings output	stdio.h
sscanf()	•	Input strings conversion	stdio.h
telldir()		Return current location	dir.h
tmpfile()	•	Create temporary binary file	stdio.h
ungetc()	•	Unget character	stdio.h
unlink()		Unlink (delete) file	modes.h
unlinkx()		Unlink (delete) file	modes.h
vfprintf()	•	Print to file	stdio.h
vprintf()	•	Print to standard output	stdio.h
vsprintf()	•	Print to string	stdio.h
write()		Write bytes to path	modes.h
writeln()		Write bytes to path	modes.h

Interprocess Communication Functions

Functions available for interprocess communication are shown in **Table 1-10**. ANSI prototypes for non-ANSI functions are achieved by defining the macro name OPT_PROTOS during the compile.

Table 1-10 Interprocess Communication Functions

Function	ANSI Standard	Description	Header File
alm_atdate()		Send signal at Gregorian date/time	alarm.h
alm_atjul()		Send signal at Julian date/time	alarm.h
alm_cycle()		Send signal at specified time intervals	alarm.h
alm_delete()		Remove pending alarm request	alarm.h
alm_set()		Send signal after specified time interval	alarm.h
_ev_creat()		Create event	events.h
_ev_delete()		Delete event	events.h
_os9_ev_info()		Returns event information in the caller's buffer	events.h
_ev_info()		Get event information	events.h

Table 1-10 Interprocess Communication Functions (continued)

Function	ANSI Standard	Description	Header File
_ev_link()		Link to existing event	events.h
_ev_pulse()		Signal event occurrence	events.h
_ev_read()		Read event without waiting	events.h
_ev_set()		Set event variable and signal event occurrence	events.h
_ev_setr()		Set relative event variable and signal event occurrence	events.h
_ev_signal()		Signal event occurrence	events.h
_ev_unlink()		Unlink event	events.h
_ev_wait()		Wait for event	events.h
_ev_waitr()		Wait for relative event	events.h
intercept()		Set up process signal handler	signal.h
_os9_alarm_atdate()		Send signal at Gregorian date/time	alarm.h
_os9_alarm_atjul()		Send signal at Julian date/time	alarm.h

Table 1-10 Interprocess Communication Functions (continued)

Function	ANSI Standard	Description	Header File
_os_alarm_atime()		Send signal at specified time	alarm.h
_os_alarm_cycle()		Send signal at specified time intervals	alarm.h
_os_alarm_delete()		Remove pending alarm request	alarm.h
_os_alarm_reset()		Reset existing alarm request	alarm.h
_os_alarm_set()		Send signal after specified time interval	alarm.h
_os_clrsigs()		Clear process signal queue	signal.h
_os_ev_allclr()		Wait for all bits defined by mask to become clear	events.h
_os_ev_allset()		Wait for event to occur	events.h
_os_ev_anyclr()		Wait for event to occur	events.h
_os_ev_anyset()		Wait for event to occur	events.h
_os_ev_change()		Wait for event to occur	events.h
_os_ev_creat()		Create new event	events.h

Table 1-10 Interprocess Communication Functions (continued)

Function	ANSI Standard	Description	Header File
`_os_ev_delete()`		Delete existing event	`events.h`
`_os_ev_info()`		Return event information	`events.h`
`_os_ev_link()`		Link to existing event by name	`events.h`
`_os_ev_pulse()`		Signal event occurrence	`events.h`
`_os_ev_read()`		Read event value without waiting	`events.h`
`_os_ev_set()`		Set event variable and signal event occurrence	`events.h`
`_os_ev_setand()`		Set event variable and signal event occurrence	`events.h`
`_os_ev_setor()`		Set event variable and signal event occurrence	`events.h`
`_os_ev_setr()`		Set relative event variable and signal event occurrence	`events.h`
`_os_ev_setxor()`		Set event variable and signal event occurrence	`events.h`
`_os_ev_signal()`		Signal event occurrence	`events.h`

Table 1-10 Interprocess Communication Functions (continued)

Function	ANSI Standard	Description	Header File
_os_ev_tstset()		Wait for event to occur	events.h
_os_ev_unlink()		Unlink event	events.h
_os_ev_wait()		Wait for event to occur	events.h
_os9_ev_wait()		Wait for event to occur	events.h
_os_ev_waitr()		Wait for relative event to occur	events.h
_os9_ev_waitr()		Wait for relative event to occur	events.h
_os_intercept()		Set up signal intercept trap	signal.h
_os9_salarm_atdate()		Execute system-state subroutine at Gregorian date/time	alarm.h
_os_salarm_atime()		Execute system-state subroutine at specified time	alarm.h
_os9_salarm_atjul()		Execute system-state subroutine at Julian date/time	alarm.h
_os_salarm_cycle()		Set cyclic alarm clock	alarm.h

Table 1-10 Interprocess Communication Functions (continued)

Function	ANSI Standard	Description	Header File
_os9_salarm_cycle()		Execute system-state subroutine every n ticks/seconds	alarm.h
_os_salarm_delete()		Remove pending alarm request	alarm.h
_os_salarm_reset()		Reset alarm clock	alarm.h
_os_salarm_set()		Set alarm clock	alarm.h
_os9_salarm_set()		Execute system-state subroutine after specified time interval	alarm.h
_os_sema_init()		Initialize semaphore data structure for use	semaphore.h
_os_sema_p()		Reserve semaphore (acquire exclusive access)	semaphore.h
_os_sema_term()		Terminate use of semaphore data structure	semaphore.h
_os_sema_v()		Release semaphore (release exclusive access)	semaphore.h
_os_send()		Send signal to another process	signal.h

Table 1-10 Interprocess Communication Functions (continued)

Function	ANSI Standard	Description	Header File
`_os_siglngj()`		Set signal mast value and return on specified stack image	`signal.h` `types.h`
`_os_sigmask()`		Mask/unmask signals during critical code	`signal.h`
`_os_sigreset()`		Reset signal intercept context stack	`signal.h`
`_os_sigrs()`		Resize current process' signal queue block	`signal.h`
`pause()`		Wait for signal	`signal.h`
`raise()`	•	Send signal to program	`signal.h`
`sigmask()`		Control process' signal handling	`signal.h`
`signal()`	•	Specify signal handling	`signal.h`

Interrupt Manipulation Functions

The following functions are provided for interrupt manipulation. ANSI prototypes for non-ANSI functions are achieved by defining the macro name OPT_PROTOS during the compile.

Table 1-11 Interrupt Manipulation Functions

Function	ANSI Standard	Description	Header File
irq_change()		Set IRQ level	regs.h
irq_disable()		Mask interrupts	regs.h
irq_enable()		Unmask interrupts	regs.h
irq_maskget()		Mask interrupts; return previous level	regs.h
irq_restore()		Restore interrupt level	regs.h
irq_save()		Return current interrupt level	regs.h
_os_firq()		Add or remove device from fast IRQ table	regs.h
_os_irq()		Add or remove device from IRQ table	regs.h
_os9_irq()		Add or remove device from IRQ table	regs.h
_os_rte()		Return from interrupt exception	signal.h

Mathematical Functions

The following are the available mathematical functions.

Table 1-12 Mathematical Functions

Function	ANSI Standard	Description	Header File
abs()	•	Integer absolute value	stdlib.h
acos()	•	Arc cosine function	math.h
asin()	•	Arc sine function	math.h
atan()	•	Arc tangent function	math.h
atan2()	•	Arc tangent function	math.h
ceil()	•	Ceiling function	math.h
cos()	•	Cosine function	math.h
cosh()	•	Hyperbolic cosine function	math.h
div()	•	Compute quotient and remainder	stdlib.h
exp()	•	Exponential function	math.h
fabs()	•	Floating absolute value	math.h
floor()	•	Floor function	math.h
fmod()	•	Compute floating point remainder	math.h

Table 1-12 Mathematical Functions (continued)

Function	ANSI Standard	Description	Header File
`frexp()`	•	Return portions of floating point number	`math.h`
`hypot()`		Euclidean distance function	`math.h`
`labs()`	•	Compute absolute value	`stdlib.h`
`ldexp()`	•	Multiply float by exponent of 2	`math.h`
`ldiv()`	•	Compute quotient and remainder	`stdlib.h`
`log()`	•	Natural logarithm	`math.h`
`log10()`	•	Base-ten logarithm	`math.h`
`modf()`	•	Return parts of real number	`math.h`
`pow()`	•	Power function	`math.h`
`rand()`	•	Return random integer	`stdlib.h`
`sin()`	•	Sine function	`math.h`
`sinh()`	•	Hyperbolic sine function	`math.h`
`sqrt()`	•	Square root function	`math.h`
`srand()`	•	Set seed for random number generator	`stdlib.h`
`tan()`	•	Tangent function	`math.h`
`tanh()`	•	Hyperbolic tangent function	`math.h`

Memory Management Functions

The memory management functions are provided for requesting and freeing memory in a machine and operating system independent manner. ANSI prototypes for non-ANSI functions are achieved by defining the macro name OPT_PROTOS during the compile.

Table 1-13 Memory Management Functions

Function	ANSI Standard	Description	Header File
calloc()	•	Allocate storage for array	stdlib.h
_cpymem()		Copy external memory	process.h
ebrk()		Get external memory	stdlib.h
free()	•	Return memory	stdlib.h
_freemem(), freemem()		Determine size of unused stack area	stdlib.h
_freemin()		Set memory reclamation bound	stdlib.h
ibrk()		Request internal memory	stdlib.h
_lcalloc()		Allocate storage for array (low-overhead)	stdlib.h
_lfree()		Return memory (low-overhead)	stdlib.h
_lmalloc()		Allocate memory from arena (low-overhead)	stdlib.h
_lrealloc()		Resize block of memory (low-overhead)	stdlib.h

Table 1-13 Memory Management Functions (continued)

Function	ANSI Standard	Description	Header File
malloc()	•	Allocate memory from arena	stdlib.h
_mallocmin()		Set minimum allocation size	stdlib.h
memchr()	•	Memory search	string.h
memcmp()	•	Compare memory	string.h
memcpy()	•	Copy memory	string.h
memmove()	•	Move memory	string.h
memset()	•	Fill memory	string.h
_os_alltsk()		Allocate task	process.h
_os_chkmem()		Check memory block's accessibility	process.h
_os_cpymem()		Copy external memory	process.h
_os_deltsk()		Deallocate process descriptor	process.h
_os_get_blkmap()		Get free memory block map	memory.h
_os_mem()		Resize data memory area	memory.h
_os_move()		Move data (low bound first)	memory.h

Table 1-13 Memory Management Functions (continued)

Function	ANSI Standard	Description	Header File
_os_permit()		Allow access to memory block	process.h
_os_protect()		Prevent access to memory block	process.h
_os_srqmem()		System memory request	memory.h
_os9_srqmem()		System memory request	memory.h
_os_srtmem()		Return system memory	memory.h
_os_transadd()		Translate memory address (OS-9)	virtual.h
_os9_translate()		Translate memory address (OS-9 for 68K)	memory.h
realloc()	•	Resize block of memory	stdlib.h
sbrk()		Extend data memory segment	stdlib.h
srqcmem()		Allocate colored memory	memory.h
_srqmem()		System memory request	memory.h
_srtmem()		System memory return	memory.h

Miscellaneous Functions

Miscellaneous functions in **Table 1-14** appear in the Ultra C libraries to relieve you from routine tasks. ANSI prototypes for non-ANSI functions are achieved by defining the macro name `OPT_PROTOS` during the compile.

Table 1-14 Miscellaneous Functions

Function	ANSI Standard	Description	Header File
`assert()`	•	Place diagnostics in programs	`assert.h`
`change_static()`		Change current static storage pointer	`regs.h`
`_cmpnam()`		Compare two strings	`strings.h`
`_errmsg()`		Print error message	`stdio.h`
`get_static()`		Return current static storage pointer	`regs.h`
`getuid()`		Determine user ID number	`process.h`
`localeconv()`	•	Numeric formatting convention inquiry	`locale.h`
`longjmp()`	•	Non-local goto	`setjmp.h`
`_os9_allbit()`		Set bits in bit map	`memory.h`
`_os_cache()`		Cache control	`cache.h`
`_os_config()`		Configure an element	`sysglob.h`
`_os_cmpnam()`		Compare two names	`strings.h`

Table 1-14 Miscellaneous Functions (continued)

Function	ANSI Standard	Description	Header File
_os9_delbit()		Deallocate in bit map	memory.h
_os_dexec()		Execute debugged program	dexec.h
_os_dexit()		Exit debugged program	dexec.h
_os9_findpd()		Find fixed block of memory	process.h
_os_getpd()		Find path descriptor	io.h
_os9_panic()		System catastrophic occurrence	process.h
_os_perr()		Print error message	errno.h
_os_prsnam()		Parse path name	strings.h
_os9_retpd()		Return fixed block of memory	process.h
_os_scache()		Cache control	regs.h
_os9_schbit()		Search bit map for free area	memory.h
_os_setsvc()		Service request table initialization	svctbl.h
_os9_setsvc()		Service request table initialization	funcs.h
_os_setuid()		Set user ID number	process.h
_os_strap()		Set error trap handler	settrap.h

Table 1-14 Miscellaneous Functions (continued)

Function	ANSI Standard	Description	Header File
_os9_strap()		Set error trap handler	settrap.h
_os_sysid()		Return system identification	regs.h
_os_uacct()		User accounting	process.h
pause()		Parse disk file (RBF) pathlist	strings.h
perror()	•	Map error number	stdio.h
prerr()		Print error message	stdio.h
_prgname()		Get module name	module.h
_prsnam()		Parse path name segment	strings.h
setjmp()	•	Non-local goto	setjmp.h
setlocale()	•	Locale control	locale.h
setuid()		Set user ID	process.h
__stacksiz(), _stacksiz()		Get size of stack used	stdlib.h
_strass()		Structure assignment	strings.h
tmpnam()	•	Generate unique valid filename	stdio.h
va_arg()	•	Get parameter in variable parameter list	stdarg.h

Table 1-14 Miscellaneous Functions (continued)

Function	ANSI Standard	Description	Header File
va_end()	•	End references to current variable parameter list	stdarg.h
va_start()	•	Initialize variable parameter list	stdarg.h

Module Manipulation Functions

Table 1-15 shows the module manipulation functions. ANSI prototypes for non-ANSI functions are achieved by defining the macro name 0PT_PR0T0S during the compile.

Table 1-15 Module Manipulation Functions

Function	ANSI Standard	Description	Header File
crc()		Calculate module CRC	module.h
_get_module_dir()		Get module directory entry	module.h
make_module()		Create module	module.h
_mkdata_module()		Create data memory module	module.h
modcload()		Load module into colored memory	module.h
modlink()		Link to memory module	module.h

Table 1-15 Module Manipulation Functions (continued)

Function	ANSI Standard	Description	Header File
modload()		Load and link to memory module	module.h
modloadp()		Load and link to memory module using PATH	module.h
munlink()		Unlink from module	module.h
munload()		Unload module	module.h
_os_altmdir()		Set alternate working module directory	moddir.h
_os_chainm()		Execute new primary module given pointer to module	process.h
_os_chmdir()		Change process' current module directory	moddir.h
_os_cmdperm()		Change permissions of module directory	moddir.h
_os_crc()		Generate CRC	module.h
_os_datmod()		Create data module	module.h
_os_delmdir()		Delete existing module directory	moddir.h
_os_fmod()		Find module directory entry	module.h

Table 1-15 Module Manipulation Functions (continued)

Function	ANSI Standard	Description	Header File
`_os_get_mdp()`		Get current and alternate module directory pathlists	`moddir.h`
`_os_get_moddir()`		Get copy of module directory	`module.h`
`_os_initdata()`		Initialize static storage from module	`module.h`
`_os_link()`		Link to memory module	`module.h`
`_os_linkm()`		Link to memory module by module pointer	`module.h`
`_os_load()`		Load module(s) from file	`module.h`
`_os_loadp()`		Load and link to a memory module using PATH	`module.h`
`_os_makmdir()`		Create new module directory	`moddir.h`
`_os_mkmodule()`		Create module of specified color type	`module.h`
`_os_modaddr()`		Find module given pointer	`module.h`
`_os_setcrc()`		Generate valid CRC in module	`module.h`
`_os_slink()`		Install subroutine module	`module.h`

Table 1-15 Module Manipulation Functions (continued)

Function	ANSI Standard	Description	Header File
_os_slinkm()		Install user subroutine library module by module pointer	module.h
_os_tlink()		Install system state trap handling module	module.h
_os_tlinkm()		Install user trap handling module by module pointer	module.h
_os_unlink()		Unlink module by address	module.h
_os_unload()		Unlink module by name	module.h
_os_vmodul()		Verify module	module.h
_os9_vmodul()		Validate module	module.h
_setcrc()		Re-validate module CRC	module.h
_subcall()		Call subroutine module	module.h

Multibyte Character and String Functions

Five functions, identified in **Table 1-16**, are available for handling multibyte characters and strings.

Table 1-16 Multi-byte Character/String Functions

Function	ANSI Standard	Description	Header File
`mblen()`	•	Determine number of bytes in multibyte character	`stdlib.h`
`mbstowcs()`	•	Convert sequence of multibyte characters	`stdlib.h`
`mbtowc()`	•	Determine number of bytes in multibyte characters	`stdlib.h`
`wcstombs()`	•	Convert sequence of wide characters to multibyte characters	`stdlib.h`
`wctomb()`	•	Convert wide character to multibyte character	`stdlib.h`

OS System Functions

The standard library provides a large number of functions directly accessing system calls. Functions available to provide access to selected system calls are identified in **Table 1-17**. ANSI prototypes for non-ANSI functions are achieved by defining the macro name `OPT_PROTOS` during the compile.

Table 1-17 OS System Functions

Function	ANSI Standard	Description	Header File
`_getsys()`		Get system global variables	`setsys.h`
`_os_sysdbg()`		Call system debugger	`process.h`
`_setsys()`		Set/examine system global variables	`setsys.h`
`_sysdbg()`		Call system debugger	`process.h`

POSIX Messaging Functions

The OS-9 implementation of the POSIX specification supports a message queue that is either visible between process or between threads. Currently, only the threaded message queue is supported. You distinguish which type you wish to create by the message queue name passed to `mq_open`. If the name starts with a '/', then a process wide message queue is assumed. Otherwise, a threaded message queue is created.

There is a thread and non-thread version of the POSIX message library. At this time, OS-9 does not support process-wide message queues. Therefore, the non-threaded version of the library is merely a placeholder until the process-wide message queues are implemented. The functions all return `ENOSYS`—the POSIX error code that means not implemented.

The supported POSIX messaging functions are listed in **Table 1-18**.

Table 1-18 POSIX Messaging Functions

Function	POSIX Standard	Description	Header File
`mq_close`	•	Close a Message Queue	`mqueue.h`
`mq_getattr`	•	Get Message Queue Attributes	`mqueue.h`
`mq_notify`	•	Notify Process That a Message is Available on a Queue	`mqueue.h`
`_mq_notify_write`		Notify Process That a Space in the Message Queue is Available	`mqueue.h`
`mq_open`	•	Open a Message Queue	`mqueue.h`

Table 1-18 POSIX Messaging Functions (continued)

Function	POSIX Standard	Description	Header File
mq_receive	•	Receive a Message From a Message Queue	mqueue.h
mq_send	•	Send a Message to a Message Queue	mqueue.h
mq_setattr	•	Set Message Queue Attributes	mqueue.h
mq_unlink	•	Remove a Message Queue	mqueue.h

Process Manipulation Functions

The standard library provides functions for manipulating processes as identified in **Table 1-19**. ANSI prototypes for non-ANSI functions are achieved by defining the macro name OPT_PROTOS during the compile.

Table 1-19 Process Manipulation Functions

Function	ANSI Standard	Description	Header File
chainc(), chain()		Load and execute new module	process.h
getpid()		Determine process ID number	process.h
_get_process_desc()		Get process descriptor copy	process.h
_get_process_table()		Get process table entry	process.h
kill()		Send signal to process	signal.h
_os9_allpd()		Allocate fixed-length block of memory	process.h
_os_alocproc()		Allocate process descriptor	process.h
_os_aproc()		Insert process in active process queue	process.h
_os9_aproc()		Enter process in active process queue	process.h
_os_chain()		Execute new primary module	process.h

Table 1-19 Process Manipulation Functions (continued)

Function	ANSI Standard	Description	Header File
_os_dfork()		Fork process under control of debugger	dexec.h
_os9_dfork()		Fork process under control of debugger	dexec.h
_os_dforkm()		Fork process under control of debugger	dexec.h
_os_exec()		Start child process	process.h
_os_findpd()		Find process descriptor	process.h
_os_fork()		Create new process	process.h
_os_forkm()		Create new process by module pointer	process.h
_os_get_prtbl()		Get copy of process descriptor block table	process.h
_os_getstat()		Get file/device status	sg_codes.h
_os_getsys()		Examine system global variable	sysglob.h
_os_gprdsc()		Get process descriptor copy	process.h
_os9_gspump()		Return data for specified process' memory map	process.h
_os_id()		Get process ID/user ID	process.h

Table 1-19 Process Manipulation Functions (continued)

Function	ANSI Standard	Description	Header File
_os9_id()		Get process ID/user ID	process.h
_os_nproc()		Start next process	process.h
_os_rtnprc()		Deallocate process descriptor	process.h
_os_setpr()		Set process priority	process.h
_os_sleep()		Put calling process to sleep	signal.h
_os9_sleep()		Put calling process to sleep	signal.h
_os_suspend()		Suspend process	process.h
_os_wait()		Wait for child process to terminate	process.h
os9exec()		OS-9 for 68K system call processing	process.h
os9fork(), os9forkc()		Create process	process.h
os9kexec()		OS-9 create process	types.h
setpr()		Set process priority	process.h
sleep()		Suspend execution for specified time	signal.h

Table 1-19 Process Manipulation Functions (continued)

Function	ANSI Standard	Description	Header File
tsleep()		Sleep for specified interval	signal.h
wait()		Wait for process termination	process.h

Resource Locks

On OS-9 for 68K systems, the functions identified in **Table 1-20** are available for use with resource locks. For more information about resource locks, refer to the ***OS-9 for 68K Technical Manual***.

Table 1-20 Resource Lock Functions

Function	ANSI Standard	Description	Header File
_os_acqlk()		Acquire ownership of resource lock	lock.h
_os_caqlk()		Conditionally acquire ownership of resource lock	lock.h
_os_crlk()		Create new resource lock descriptor	lock.h
_os_ddlk()		Check for deadlock situation	process.h
_os_dellk()		Delete existing lock descriptor	lock.h

Table 1-20 Resource Lock Functions (continued)

Function	ANSI Standard	Description	Header File
`_os_rellk()`		Release ownership of resource lock	`lock.h`
`_os_waitlk()`		Activate next process waiting to acquire lock	`lock.h`

Searching and Sorting Functions

Functions included for searching and sorting are identified in **Table 1-21**.

Table 1-21 Search and Sort Functions

Function	ANSI Standard	Description	Header File
`bsearch()`	•	Search array	`stdlib.h`
`qsort()`	•	Quick sort	`stdlib.h`

String Handling Functions

The C language does not have a character string data type. Instead, it stores strings as character arrays and the standard library provides functions to manipulate them. The functions are identified in **Table 1-22**. ANSI prototypes for non-ANSI functions are achieved by defining the macro name `OPT_PROTOS` during the compile.

Table 1-22 String Handling Functions

Function	ANSI Standard	Description	Header File
`findnstr()`		Search string for pattern	`strings.h`
`findstr()`		Search string for pattern	`strings.h`
`index()`		Search for character in string	`strings.h`
`rindex()`		Search for character in string	`strings.h`
`strcat()`	•	String catenation	`string.h`
`strchr()`	•	Locate string	`string.h`
`strcmp()`	•	String comparison	`string.h`
`strcoll()`	•	String comparison	`string.h`
`strcpy()`	•	String copy	`string.h`
`strcspn()`	•	Get string length	`string.h`
`strerror()`	•	Map error message string	`string.h`
`strftime()`	•	Place formatted time in buffer	`time.h`
`strhcpy()`		Copy old OS-9 for 68K strings	`strings.h`

Table 1-22 String Handling Functions (continued)

Function	ANSI Standard	Description	Header File
strlen()	•	Determine string length	string.h
strncat()	•	String catenation	string.h
strncmp()	•	String comparison	string.h
strncpy()	•	String copy	string.h
strpbrk()	•	Locate first occurrence of string	string.h
strrchr()	•	Locate last occurrence of string	string.h
strspn()	•	Compute string length	string.h
strstr()	•	Locate first occurrence of string	string.h
strtod()	•	String to double conversion	stdlib.h
strtok()	•	Break string into tokens	string.h
strtol()	•	String to long conversion	stdlib.h
strtoul()	•	String to unsigned long conversion	stdlib.h
strxfrm()	•	Transform string	string.h

Terminal Manipulation Functions (Termcap)

Table 1-23 Termcap Functions

Function	ANSI Standard	Description	Header File
`tgetent()`		Get termcap entries	`termcap.h`
`tgetflag()`		Check terminal capability presence	`termcap.h`
`tgetnum()`		Get terminal capability ID	`termcap.h`
`tgetstr()`		Get terminal capability	`termcap.h`
`tgoto()`		Get cursor movement capability	`termcap.h`
`tputs()`		Output capability string	`termcap.h`

Time Functions

The TZ environment variable is used to specify the time zone for the C functions to use. TZ should have the following format:

```
zzz[+/-n][:ddd]
```

zzz is one of the supported time zone names listed in **Table 1-24**.

Table 1-24 Time Zones and Descriptions

Zone	Description
GMT/UTC	Greenwich Mean Time (or Coordinated Universal Time)
PST/PDT	USA Pacific Standard Time/Daylight Savings Time
MST/MDT	USA Mountain Standard Time/Daylight Savings Time
CST/CDT	USA Central Standard Time/Daylight Savings Time
EST/EDT	USA Eastern Standard Time/Daylight Savings Time
YST	Yukon Standard Time (Most of Alaska)
AST	Aleutian/Hawaiian Standard Time
EET	Eastern European Time
CET	Central European Time
WET	Western European Time
JST	Japan Standard Time
MIT	Midway Islands Time
HST	Hawaii Standard Time

Table 1-24 Time Zones and Descriptions (continued)

Zone	Description
PNT	Phoenix Standard Time
IET	Indiana Eastern Standard Time
PRT	Puerto Rico Standard Time
CNT	Canada Newfoundland Time
AGT	Argentina Standard Time
BEZ	Brazillian Standard Time
CAT	Central Africa
ECT	European Central Time
ART	Arabic Standard Time
EAT	Eastern African Time
MET	Middle Eastern Time
NET	Near East Time
PLT	Pakistan Lahore Time
IST	India Standard Time
BST	Bangladesh Standard Time
VST	Vietnam Standard Time
CTT	China Taiwan Standard Time

Table 1-24 Time Zones and Descriptions (continued)

Zone	Description
ACT	Australia Central Time
AET	Australian Eastern Time
SST	Solomon Standard Time
NST	New Zealand Standard Time

n is the optional number of minutes east (+) or west (-) of the time zone.

ddd is the option controlling Daylight Savings Time.

Recognized codes for ddd are shown in **Table 1-25**.

Table 1-25 Valid ddd Codes

Value	Description
no	Do not use Daylight Savings Time
usa	Conforms to the US Uniform Time Act of 1967 and its various amendments through 1987
eur	Observe European Daylight Savings Time

For example, the following command specifies that the compiler should use the USA Eastern Daylight Savings Time and should conform to the US Uniform Time Act of 1967:

```
setenv TZ EDT:usa
```

The American time zones default to usa, with the exception of AST which defaults to no. The European time zones default to eur and GMT/UTC defaults to no.

If the TZ environment variable is not set and, if on OS-9, m_tmzone is not set, time() uses USA Central Standard Time (CST).

Ultra C provides time functions identified in **Table 1-26**. ANSI prototypes for non-ANSI functions are achieved by defining the macro name OPT_PROTOS during the compile.

Table 1-26 Time Functions

Function	ANSI Standard	Description	Header File
asctime()	•	Convert broken-down time to string format	time.h
clock()	•	Get processor time	time.h
ctime()	•	Convert calendar time to string format	time.h
difftime()	•	Find temporal difference	time.h
getime()	•	Get system time	time.h
gmtime()	•	Convert calendar time to Greenwich Mean Time	time.h
_gregorian()		Convert date/time to Gregorian value	time.h
_julian()		Convert date/time to Julian value	time.h
localtime()	•	Convert calendar time to local time	time.h
mktime()	•	Convert broken-down time to calendar time	time.h
_os_getime()		Get system date and time	time.h

Table 1-26 Time Functions (continued)

Function	ANSI Standard	Description	Header File
`_os9_getime()`		Get system date/time	`time.h`
`_os_gregorian()`		Get Gregorian date	`time.h`
`_os_julian()`		Get Julian date	`time.h`
`_os_setime()`		Set system date/time	`time.h`
`_os9_setime()`		Set system date/time	`time.h`
`setime()`		Set system time	`time.h`
`strftime()`	•	Place formatted time in buffer	`time.h`
`sys_mktime()`		Convert local time to Greenwich Mean Time	`time.h`
`_sysdate()`		Get current system date/time	`time.h`
`time()`	•	Get calendar time	`time.h`

Processor-Specific Functions

Functions available only on the 80x86 family of systems are identified in **Table 1-27**. ANSI prototypes for non-ANSI functions are achieved by defining the macro name `OPT_PROTOS` during the compile.

Table 1-27 80x86-Only Functions

Function	ANSI Standard	Description	Header File
`get_current_tss()`		Get current task segment	`regs.h`
`get_excpt_base()`		Return exception table base address	`regs.h`
`get_gdtr()`		Get global descriptor pointer	`regs.h`
`make_gdesc()`		Make global descriptor table entry	`regs.h`
`make_idesc()`		Make interrupt descriptor table entry	`regs.h`
`set_excpt_base()`		Set exception table base address	`regs.h`
`set_gdtr()`		Set global descriptor pointer	`regs.h`

These functions are available only on the PowerPC processors as identified in **Table 1-28**. ANSI prototypes for non-ANSI functions are achieved by defining the macro name `OPT_PROTOS` during the compile.

Table 1-28 Power PC-Only Functions

Function	ANSI Standard	Description	Header File
`_get_<name>()`		Read the value of SPRs, DCRs, PREGs, and TBs	`getset.h`
`_set_<name>()`		Set the value of SPRs, DCRs, PREGs, and TBs	`getset.h`

Implementation-Defined Behavior

The following are the implementation-defined issues pertaining to the Ultra C library. Each bulleted item contains one implementation-defined issue. The number in parentheses included with each bulleted item indicates the location in the ANSI specification where you can find more information.

- The null pointer constant to which the macro NULL expands (4.1.5).

  NULL expands to `(void *) 0`.

- The diagnostic printed by and the termination behavior of the `assert()` function (4.2).

  `assert()` prints a line with the following syntax (if the `NDEBUG` macro is not defined) if the assertion fails:

  ```
  Assertion failed:<expression>, file <file>, line <line>\n
  ```

  `<expression>` is the expression being tested.

  `<file>` is the file name of the file containing the `assert` macro.

  `<line>` is the line number of the file containing the `assert` macro.

  The `abort()` function is called after the assertion failure line is printed. This raises the `SIGABRT` signal that may be handled by the application.

For More Information

Refer to the description of `signal()` for more information about `SIGABRT`.

- The sets of characters tested for by the `isalnum()`, `isalpha()`, `iscntrl()`, `islower()`, `isprint()`, and `isupper()` functions (4.3.1).

  `isalnum()` returns non-zero for any character which `isdigit()` or `isalpha()` returns non-zero.

  `isalpha()` returns non-zero for any character which `islower()` or `isupper()` returns non-zero.

  `iscntrl()` returns non-zero for ASCII characters with values from 0 to 0x1f and 0x7f.

  `islower()` returns non-zero for ASCII characters with values from 0x61 to 0x7a.

  `isprint()` returns non-zero for ASCII characters with values from 0x20 to 0x7e.

  `isupper()` returns non-zero for ASCII characters with values from 0x41 to 0x5a.

- The values returned by the mathematics functions on domain errors (4.5.1).

  The mathematics functions return HUGE_VAL on domain errors.

- Whether the mathematics functions set the integer expression `errno` to the value of the macro ERANGE on underflow range errors (4.5.1).

  `errno` is not set to `ERANGE` on underflow errors.

- Whether a domain error occurs or zero is returned when the `fmod()` function has a second argument of zero (4.5.6.4).

  `fmod()` returns zero if it has a second argument of zero.

- The set of signals for the `signal()` function (4.7.1.1).

  The set of signals for the signal function varies with the operating system and the processor. A set of signals is also available regardless of operating system.

ANSI “Core” Signals

```
SIGABRT     SIGFPE     SIGILL     SIGINT
SIGSEGV     SIGTERM
```

OS-9 “Core” Signals

```
SIGKILL     SIGWAKE    SIGQUIT    SIGHUP
SIGALRM     SIGPIPE    SIGUSR1    SIGUSR2
```

OS-9 68K Family Signals

```
SIGADDR     SIGCHK     SIGTRAPV   SIGPRIV
SIGTRACE    SIG1010    SIG1111
```

OS-9 80x86 Family Signals

```
SIGGPROT    SIGSTACK   SIGSEGNP   SIGINVTSS
SIGDBLFLT   SIGBNDCHK  SIGBRKPT   SIGNMI
SIGDBG
```

OS-9 PowerPC™ Family Signals

SIGCHK SIGINST SIGPRIV SIGALIGN

- The semantics for each signal recognized by the `signal()` function (4.7.1.1).

For More Information

For information related to the exceptions, see the reference manual for your specific processor supplied by the manufacturer.

Table 1-29 Signals

Signal	Condition
SIGABRT	Abnormal termination such as initiated by the `abort()` function.
SIGFPE	An erroneous arithmetic operation such as zero divide or an operation resulting in overflow. This signal is generated on target-specific conditions.
SIGILL	Detection of an invalid function image such as illegal instruction. This signal is generated on target-specific conditions.
SIGINT	Receipt of an interactive attention signal generated by the keyboard interrupt character on OS-9 for 68K and OS-9.
SIGSEGV	An invalid access to storage. This signal is generated on target-specific conditions.
SIGTERM	A termination request sent to the program.

Table 1-30 More Signals

Signal	Condition
SIGKILL	A death request sent to the program. This signal cannot be ignored or handled.
SIGWAKE	A wakeup request sent to the program. If a process is active when this signal arrives, it is ignored. Otherwise, the process is activated, but the signal handling function is not called.
SIGQUIT	Receipt of an interactive abort signal generated by the keyboard quit character on OS-9.
SIGHUP	Modem hangup signal. This signal is sent to a process when a modem connection is lost.
SIGALRM	Not automatically sent. Programs can use this signal for an alarm call.
SIGPIPE	Not automatically sent.
SIGUSR1	Condition is user defined.
SIGUSR2	Condition is user defined.

Processor-Specific Family Signals

For More Information

For information related to the exceptions, see the reference manual for your specific processor.

68K Family Processors

The OS-9 68K family signals sent when their associated exception occurs are identified in **Table 1-31**.

Table 1-31 OS-9 68K Family Exception Signals

Signal	Exception
SIGADDR	Address Error
SIGCHK	CHK Instruction
SIGTRAPV	TRAPV Instruction
SIGPRIV	Privilege Violation
SIGTRACE	Trace
SIG1010	Line 1010 Emulation
SIG1111	Line 1111 Emulation

80x86 Family Processors

The OS-9 80x86 family signals shown in **Table 1-32** are sent when their associated exception occurs.

Table 1-32 OS-9 80x86 Family Exception Signals

Signal	Exception
SIGGPROT	General Protection
SIGSTACK	Stack Exception
SIGSEGNP	Segment Not Present
SIGINVTSS	Invalid TSS
SIGDBLFLT	Double Fault
SIGBNDCHK	Bounds Check
SIGBRKPT	Breakpoint
SIGNMI	Non-Maskable Interrupt
SIGDBG	Debug Exceptions

PowerPC Family Processors

The OS-9 PowerPC family signals shown in **Table 1-33** are sent when their associated exception occurs.

Table 1-33 OS-9 PowerPC Family Exception Signals

Signal	Exception
SIGCHECK	Machine Check
SIGINST	Instruction Access

Table 1-33 OS-9 PowerPC Family Exception Signals

Signal	Exception
SIGPRIV	Privilege Violation
SIGALIGN	Alignment

ARM Family Processors

The OS-9 ARM family signal shown in **Table 1-34** is sent when its associated exception occurs.

Table 1-34 OS-9 ARM Family Exception Signals

Signal	Exception
SIGALIGN	Alignment

SH Family Processors

The OS-9 SH3 family signal shown in **Table 1-35** is sent when the associated exception occurs.

Table 1-35 OS-9 SH3 Family Exception Signals

Signal	Exception
SIGALIGN	Alignment

SPARC Family Processors

The OS-9 SPARC family signals shown in **Table 1-36** are sent when their associated exception occurs.

Table 1-36 OS-9 SPARC Family Exception Signals

Signal	Exception
SIGALIGN	Alignment
SIGWINDOWOV	Window Overflow
SIGWINDOWUV	Window Underflow
SIGATAGOV	Tag Overflow
SIGCPE	Coprocessor Exception

- The default handling and the handling at program startup for each signal recognized by the `signal()` function (4.7.1.1).

  For each signal recognized by `signal()`, the default handling is program termination and the program start-up condition is `SIG_DFL`. The program terminates with an exit status related to the exception or the signal number for non-exception related signals.

- If the equivalent of `signal` (`sig`, `SIG_DFL`); is not executed prior to the call of the signal handler, the blocking of the signal that is performed (4.7.1.1).

  `signal (sig, SIG_DFL)` is executed before the call to the signal handler. Signals are blocked for the duration of the signal handler, unless the program takes some action that unblocks `signal()`. Refer to your operating system technical manual for more information.

- Whether the default handling is reset if the `SIGILL` signal is received by a handler specified to the signal function (4.7.1.1).

  The default handling is reset on all signals.

- Whether the last line of a text stream requires a terminating newline character (4.9.2).

  A terminating newline is not required as the last character of a text stream.

- Whether space characters that are written out to a text stream immediately before a newline character appear when read in (4.9.2).

  Space characters written to a text stream immediately before a newline character appears when read.

- The number of null characters that may be appended to data written to a binary stream (4.9.2).

  Null characters are not appended to data written to a binary stream.

- Whether the file position indicator of an append mode stream is initially positioned at the beginning or end of the file (4.9.3).

  The file position indicator of an append mode stream is initially positioned at the beginning of the file. Regardless, all writes to a stream in append mode are started at the current end-of-file.

- Whether a write on a text stream causes the associated file to be truncated beyond that point (4.9.3).

  No truncation of this sort occurs.

- The characteristics of file buffering (4.9.3).

  The file buffering decision is made when the first request is made to the stream. If the buffering type and size have not been set explicitly by `setvbuf()`, the determination is made based on the device with which the stream is associated. If the device is interactive, line buffering is used. Otherwise, full buffering is used. If the device is RBF (Random Block File Manager), the buffer size is `2 * current_sector_size`.

- Whether a zero-length file actually exists (4.9.3).

  A zero length file actually does exist.

- The rules for composing valid file names (4.9.3).

  The rules vary depending on the operating system:

  OS-9 for 68K File names can contain 1 to 28 upper or lower case letters, digits, underscores (_), periods (.), or dollar signs ($).

  OS-9 File names can contain 1 to 43 upper or lower case letters, digits, underscores (_), periods (.), or dollar signs ($).

- Whether the same file can be open multiple times (4.9.3).

  A program can open a file more than once, but fully buffered reading and writing to the same file has undefined results.

- The effect of the `remove()` function on an open file (4.9.4.1).

  `remove()` returns -1 with `errno` set to `EOS_SHARE` if called with the name of an open file.

- The effect if a file with the new name exists prior to a call to the `rename()` function (4.9.4.2).

  Assuming the calling program has write permission, the file with the new name is deleted and recreated with the new information.

- The output for %p conversion in the `fprintf()` function (4.9.6.1).

  The pointer to `void` is printed as an unsigned hexadecimal integer.

- The input for %p conversion in the `fscanf()` function (4.9.6.2).

  The input for %p conversion is an unsigned hexadecimal integer.

- The interpretation of a '-' character that is neither the first nor the last character in the scanlist for %[conversion in the `fscanf()` function (4.9.6.2).

  A hyphen (-) character is interpreted simply as the addition of - to the scanlist.

- The value to which the macro `errno` is set by the `fgetpos()` or `ftell()` function on failure (4.9.9.1, 4.9.9.4).

  `errno` is set to the operating system specific error that is returned by either `_os_gs_size()` (in append mode) or `_os_gs_pos()`.

- The messages generated by the `perror()` function (4.9.10.4).

  The messages generated by the `perror()` function are located in the system `errmsg` file (`/dd/SYS/errmsg, /h0/SYS/errmsg`, or `/d0/SYS/errmsg`). Refer to your operating system technical manual for detailed information. Error messages are generally in the form:

  ```
  <message>: #<major>:<minor> <detail>
  ```

  `<message>` is the user-passed message string.

  `<major>` is the family of error.

  `<minor>` is the error number within the family.

  `<detail>` is the information from the system `errmsg` file (if available).

- The behavior of the `calloc()`, `malloc()`, or `realloc()` function if the size requested is zero (4.10.3).

  The `calloc()`, `malloc()`, and `realloc()` functions return `NULL` if the requested size is zero.

- The behavior of the `abort()` function with regard to open and temporary files (4.10.4.1).

  If the `SIGABRT` signal handler returns, the program is terminated abnormally flushing open streams, closing open streams, and removing temporary files.

- The status returned by the `exit()` function if the value of the argument is other than zero, `EXIT_SUCCESS`, or `EXIT_FAILURE` (4.10.4.3).

  The integer value passed to `exit()` is the exit status of the program.

- The set of environment names and the method for altering the environment list used by the `getenv()` function (4.10.4.3).

  The common set of environment names depends on the operating system. Refer to your operating system's user manual for more information.

  The environment list is implemented as a NULL terminated array of pointers to characters. Each string is in the form:

  `<name>=<value>`

  `<name>` is the name of the environment variable.

  `<value>` is the current value of the environment variable.

  To modify the list, you must either find the existing entry or enlarge the array and add an entry. The base of the array is stored in the external variable `_environ`.

- The contents and mode of execution of the string by the `system()` function (4.10.4.5).

  The contents of an execution string given to `system()` can be any valid shell command line. The shell used to execute the string is either the current value of the `SHELL` environment variable or `shell` if `SHELL` is unavailable.

- The contents of the error message strings returned by the `strerror()` function (4.11.6.2).

  The strings generated by the `strerror()` function are located in the system `errmsg` file (`/dd/SYS/errmsg, /h0/SYS/errmsg,` or `/d0/SYS/errmsg`). Refer to your operating system technical manual for detailed information. Error messages are generally in the form:

#<major>:<minor> <detail>

<major> is the family of error.

<minor> is the error number within the family.

<detail> is the information from the system errmsg file (if available).

- The local time zone and Daylight Savings Time (4.12.1).

  The local time zone and Daylight Savings Time are determined by the TZ environment variable on OS-9 for 68K and the TZ variable or the m_tmzone field in the init module on OS-9. Failing to find user defined values, the defaults are Central Standard Time and Central Daylight Time.

- The era for the clock() function (4.12.2.1).

  The era for clock() is the elapsed ticks since the program was forked.

Equivalents for the sys_clib.l Functions

The functions in the sys_clib.l library exist mainly for compatibility with the Microware K & R C compiler, although the calls are accepted in all modes. Whenever possible, you should use the functions in the os_lib.l and clib.l libraries that have the same functionality. The following is a list of the sys_clib.l functions and the corresponding calls in either os_lib.l or clib.l.

Table 1-37 sys_clib.l Function Equivalents

sys_clib.l Function	Corresponding Function
access()	_os_open()
create()	_os_create()
alm_atdate()	_os_alarm_atime()/ _os9_alarm_atdate()
detach()	_os_detach()

Table 1-37 sys_clib.l Function Equivalents (continued)

sys_clib.l Function	Corresponding Function
alm_atjul()	_os_alarm_atime()/ _os9_alarm_atjul()
dup()	_os_dup()
alm_cycle()	_os_alarm_cycle()
ebrk()	malloc()
alm_delete()	_os_alarm_delete()
_errmsg()	strerror()
alm_set()	_os_alarm_set()
_ev_creat()	_os_ev_creat()
attach()	_os_attach()
_ev_delete()	_os_ev_delete()
_ev_info()	_os_ev_info()
chainc(), chain()	_os_chain()
_ev_link()	_os_ev_link()
chdir()	_os_chdir()
_ev_pulse()	_os_ev_pulse()
chmod()	_os_ss_attr()
_ev_read()	_os_ev_read()
chown()	_os_ss_fd()

Table 1-37 sys_clib.l Function Equivalents (continued)

sys_clib.l Function	Corresponding Function
_ev_set()	_os_ev_set()
chxdir()	_os_chdir()
_ev_setr()	_os_ev_setr()
_cleareof(), cleareof()	clearerr()
_ev_signal()	_os_ev_signal()
_ev_unlink()	_os_ev_unlink()
close()	_os_close()
_ev_wait()	_os_ev_wait()/ _os9_ev_wait()
closedir()	_os_close()
_ev_waitr()	_os_ev_waitr()/ _os9_ev_waitr()
_cmpnam()	_os_cmpnam()
_exit()	_os_exit()
_cpymem()	_os_cpymem()
fdopen()	none
crc()	_os_crc()
findnstr()	strstr()
creat()	_os_create()
findstr()	strstr()

Table 1-37 sys_clib.l Function Equivalents (continued)

sys_clib.l Function	Corresponding Function
_get_module_dir()	_os_get_moddir()
makdir()	_os_makdir()
_get_process_desc()	_os_gprdsc()
make_module()	_os_mkmodule()
_get_process_table()	_os_get_prtbl()
_mkdata_module()	_os_datmod()
mktime()	_os_getime()/ _os9_getime()
mknod()	_os_makdir()
getpid()	_os_id()
mktemp()	tmpnam()
getstat()	_os_getstat()
modcload()	_os_loadp()
_getsys()	_os_getsys()
modlink()	_os_link()
getuid()	_os_id()
modload()	_os_load()
getw()	getc()getw() is replaced by performing two calls to getc()
modloadp()	_os_loadp()

Table 1-37 sys_clib.l Function Equivalents (continued)

sys_clib.l Function	Corresponding Function
_gregorian()	_os_gregorian()
munlink()	_os_unlink()
_gs_devn()	_os_gs_devnm()
munload()	_os_unload()
_gs_eof()	_os_gs_eof()
open()	_os_open()
_gs_gfd()	_os_gs_fd()
opendir()	_os_open()
_gs_opt()	_os_gs_popt()
os9exec()	_os_exec()
_gs_pos()	_os_gs_pos()
os9fork(), os9forkc()	_os_fork()
_gs_rdy()	_os_gs_ready()
_gs_size()	_os_gs_size()
os9kexec()	_os_exec()
hypot()	none
pause()	_os_prsnam()
ibrk()	malloc()
pause()	sleep()

Table 1-37 sys_clib.l Function Equivalents (continued)

sys_clib.l Function	Corresponding Function
index()	strchr()
pffinit()	none
intercept()	_os_intercept()
pflinit()	none
_julian()	_os_julian()
prerr()	strerror()
kill()	_os_send()
_prgname()	argv[0]
lseek()	_os_seek()
_prsnam()	_os_prsnam()
putw()	putc() putc(), putw() is replaced by performing two calls to putc()
_ss_enrts()	_os_ss_enrts()
read()	_os_read()
_ss_lock()	_os_ss_lock()
readdir()	_os_read()
_ss_opt()	_os_ss_popt()
readln()	_os_readln()
_ss_pfd()	_os_ss_fd()

Table 1-37 sys_clib.l Function Equivalents (continued)

sys_clib.l Function	Corresponding Function
rewinddir()	_os_seek()
_ss_rel()	_os_ss_relea()
rindex()	strrchr()
_ss_rest()	_os_ss_reset()
sbrk()	malloc()
_ss_size()	_os_ss_size()
seekdir()	_os_seek()
_ss_ssig()	_os_ss_sendsig()
_setcrc()	_os_setcrc()
_ss_tiks()	_os_ss_ticks()
setime()	_os_setime()/ _os9_setime()
_ss_wtrk()	_os_ss_wtrack()
setpr()	_os_setpr()
_strass()	memcpy() on structure assignment
setstat()	_os_setstat()
strhcpy()	none
_setsys()	_os_setsys()
sys_mktime()	mktime() for user programs

Table 1-37 sys_clib.l Function Equivalents (continued)

sys_clib.l Function	Corresponding Function
setuid()	_os_setuid()
_sysdate()	_os_setime()/ _os9_setime()
sigmask()	_os_sigmask()
_sysdbg()	_os_sysdbg()
sleep()	_os_sleep()/ _os9_sleep()
telldir()	_os_gs_pos()
srqcmem()	_os_srqmem()
tsleep()	_os_sleep()
_srqmem()	_os_srqmem()/ _os9_srqmem()
unlink()	_os_delete()
_srtmem()	_os_srtmem()
unlinkx()	_os_delete()
_ss_attr()	_os_ss_attr()
wait()	_os_wait()
_ss_dcoff()	_os_ss_dcoff()
write()	_os_write()
_ss_dcon()	_os_ss_dcon()
writeln()	_os_writeln()

Table 1-37 sys_clib.l Function Equivalents (continued)

sys_clib.l Function	Corresponding Function
`_ss_dsrts()`	`_os_ss_dsrts()`
-	-

The Function Description

Each function description includes a minimum of the following sections:

- Syntax
- OS-9 Attributes
- Description
- Library

In addition, descriptions may also contain sections for the following:

- Examples
- Errors
- References to other calls (See Also)

Syntax

The syntax shows how the function and parameters look if written as a C function definition, even if the actual function is a macro or is written in assembly language.

For example, the syntax for `fopen()` appears as follows:

```
#include <stdio.h>
FILE *fopen(const char *name, const char *action);
```

This indicates that `fopen()` requires the `<stdio.h>` header file, returns a pointer to a structure of type `FILE`, and requires two parameters, both pointers to constant character strings. The parameter names are suggestions only; you can use any name.

OS-9 Attributes

The OS-9 Attributes section lists various attributes of each function in relation to OS-9—including whether the function is compatible with OS-9 and/or OS-9 for 68K; whether the function is in user state and/or system state; whether the function is safe for use in a threaded application; and whether the function is re-entrant.

Library

The library field indicates the libraries that contain the function.

Errors

When an error occurs, C functions typically return an error code in the global variable `errno`. You must include the file `<errno.h>` in C programs to declare `errno`.

References to Other Calls

Many functions also have a “See Also” section. Functions listed in the “See Also” section are related functions or are called by the function being described.

Chapter 2: Functions

This chapter includes library function definitions in alphabetical order according to some special rules.

- Special characters (not letters, numbers, or underscores) are listed first.
- Function calls are listed in alphabetic order next without regard for numbers and underscores.
- If two function calls are identical using these rules, then they are alphabetized according to the following order:

  1. Symbols
  2. Underscores
  3. Alphabetic characters
  4. Numbers

abort()

Abnormal Program Termination

Syntax

```
#include <stdlib.h>
void abort(void);
```

Description

`abort()` abnormally terminates a program by raising the signal `SIGABRT`. However, if the signal `SIGABRT` is being caught and the signal handler does not return, the program is not terminated. If `SIGABRT` returns, the program is terminated abnormally. This flushes and closes the open streams and removes temporary files. The process returns the status `EXIT_FAILURE`.

Note

`abort()` does not return to the caller.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

abs()

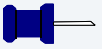
Integer Absolute Value

Syntax

```
#include <stdlib.h>
int abs(int value);
```

Description

`abs()` returns the absolute value of its integer parameter. `value` is the integer parameter. `abs(0x80000000)` returns `0x80000000` as the result.



Note

Applying `abs()` to the most negative integer yields a result which is the most negative integer:

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

MICROWARE™

access()

Determine File Accessibility

Syntax

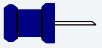
```
#include <modes.h>
int access(
     char         *name,
     int          perm);
```

Description

`access()` returns zero if the mode(s) specified in the file permissions are correct for the user to access the specified file.

`name` is a pointer to the name of the file.

`perm` may be any legal mode as defined in the `modes.h` header file. Use a mode value of zero to verify that a file exists. Only the lower 16 bits of `perm` are used. If the file cannot be accessed, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

The `perm` value may not be compatible with other systems.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`sys_clib.l`

acos()

Arc Cosine Function

Syntax

```
#include <math.h>
double acos(double x);
```

Description

`acos()` returns the arc cosine of `x`, in the range of [`0, π`] radians. A domain error, `EDOM` stored in `errno`, occurs for parameters not in the range [`-1, +1`]. On error, `acos()` returns `HUGE_VAL`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Error

`EDOM`

alarm()

Sends signal SIGALRM

Syntax

```
#include <UNIX/os9def.h>
unsigned int alarm(unsigned int seconds);
```

Description

`alarm()` requests that a signal be sent to the requesting process after a specified number (`seconds`) of seconds. Unless caught or ignored, the signal terminates the process. `alarm()` sends signal `SIGALRM` when the alarm expires.

`alarm()` requests are not stacked; successive calls reset the alarm clock. If the argument is `0`, any `alarm()` request is canceled.

`alarm()` returns `-1` and sets the global value `errno` if an error occurs.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

See Also

`_os9_alarm_atdate()`
`_os_alarm_set()`

alloca()

Allocates Bytes of Space in Stack Frame

Syntax

```
#include <UNIX/os9def.h>
void *alloca(unsigned int size);
```

Description

`alloca()` allocates `size` number of zero bytes from the `malloc()` pool. This space is not automatically freed.

`alloca()` returns a pointer to the allocated bytes. If an error occurs, `NULL` is returned and the global variable `errno` is set.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

MICROWARE™

alm_atdate()

Set Alarm at Gregorian Date/Time

Syntax

```
#include <alarm.h>

int alm_atdate(
     int          sigcode,
     int          time,
     int          date);
```

Description

`alm_atdate()` requests that a signal be sent to the requesting process at a specific Gregorian time and date. The time and date must be in the following format:

Figure 2-1 alm_atdate() Time and Date Format

	Byte 0	Byte 1	Byte 2	Byte 3
Time:	0	Hour (0-23)	Minute	Second
Date:	Year (two bytes)		Month	Day

`sigcode` specifies the signal to be sent to the caller.

`time` specifies the time to send the signal.

`date` specifies the Gregorian date for the signal to be sent.

Because the system time and date may be changed, the alarm signal is sent when the system time and date become greater than or equal to the alarm time.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

If errors do not occur, `alm_atdate()` returns the alarm ID.

For More Information

Refer to your operating system technical manual for information about alarms.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

`_os9_alarm_atdate()`
`_os_alarm_set()`

MICROWARE™

alm_atjul()

Set Alarm at Julian Date/Time

Syntax

```
#include <alarm.h>

int alm_atjul(
     int          sigcode,
     int          time,
     int          date);
```

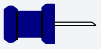
Description

`alm_atjul()` requests that a signal be sent to the requesting process at a specific Julian date and time.

`sigcode` specifies the signal to be sent to the caller.

`time` should contain the time at which to send the signal, expressed as the number of seconds after midnight.

`date` must contain the Julian date on which to send the signal. The Julian date is the number of days since 1 January 4713 B.C., the beginning of the Julian period.



Note

A Julian day begins at midnight on OS-9 systems. Standard Julian days begin twelve hours earlier, at noon. For example, 1:00 a.m. January 2, 4713 B.C. is one hour after the beginning of Julian Day 1 for OS-9, but thirteen hours after the beginning in standard Julian time.

Because the system time and date may be changed, the alarm signal is sent when the system time and date become greater than or equal to the alarm time.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

If errors do not occur, `alm_atjul()` returns the alarm ID.

For More Information

Refer to your operating system technical manual for information about alarms.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`sys_clib.l`

See Also

`_os9_alarm_atjul()`
`_os_alarm_set()`

alm_cycle()

Set Alarm at Specified Time Intervals

Syntax

```
#include <alarm.h>

int alm_cycle(
     int          sigcode,
     int          time_interval);
```

Description

`alm_cycle()` sends a signal to the requesting process after the specified time has elapsed and then resets the alarm to provide a recurring periodic signal.

`sigcode` specifies the signal to be sent to the caller.

`time_interval` specifies the periodic interval at which the signal is to be sent.

For example, if the request is made at time `X` and `time_interval` is `t`, the signal is sent to the requesting process at times `(X + t)`, `(X + 2t)`, `(X + 3t)`, and so forth until the alarm is deleted.

If the most significant bit of `time_interval` is set, `time_interval` is assumed to be in 256ths of a second. Otherwise, `time_interval` is assumed to be in units of system clock ticks (refer to `CLOCKS_PER_SEC` in the `time.h` header file). The minimum `time_interval` allowed is two system clock ticks.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

If errors do not occur, `alm_cycle()` returns the alarm ID.

For More Information

Refer to your operating system technical manual for information about alarms.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`alm_delete()`
`alm_set()`
`alm_cycle()`
`alm_delete()`
`_os_alarm_set()`

MICROWARE™

alm_delete()

Remove Pending Alarm Request

Syntax

```
#include <alarm.h>

int alm_delete(int alarmid);
```

Description

`alm_delete()` cancels the specified alarm request.

`alarmid` is the alarm ID of the alarm to cancel. If `alarmid` is zero, all pending alarm requests are cancelled.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

If errors do not occur, `alm_delete()` returns zero.

For More Information

Refer to your operating system technical manual for information about alarms.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_alarm_delete()`

alm_set()

Set Alarm after Specified Time Interval

Syntax

```
#include <alarm.h>

int alm_set(
     int          sigcode,
     int          time);
```

Description

`alm_set()` requests that a signal be sent to the requesting process after the specified time has elapsed. For example, if the request is made at time `X` and `time` is `t`, the signal is sent at time `(X + t)`, unless the alarm request is cancelled.

`sigcode` is the signal code to be sent to the requesting process.

`time` specifies the time interval. If the most significant bit of `time` is set, `time` is assumed to be in 256ths of a second. Otherwise, `time` is assumed to be in units of system clock ticks (refer to `CLOCKS_PER_SEC` in the `time.h` header file). The minimum `time` allowed is two system clock ticks.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

If errors do not occur, `alm_set()` returns the alarm ID.

For More Information

Refer to your operating system technical manual for information about alarms.

Attributes

Operating System: OS-9 and OS-9 for 68K

State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_alarm_set()`

asctime()

Convert Broken-Down Time to String Format

Syntax

```
#include <time.h>
char *asctime(struct tm *tp);
```

Description

`asctime()` converts the broken-down time structure into the following 26-byte string format, including the terminating `\0`:

`xxx mmm dd hh:mm:ss yyyy\n\0`

`xxx` is one of the following days of the week:

Table 2-1 Days of Week

Sun	Mon	Tue	Wed	Thu	Fri	Sat

`mmm` is one of the following months of the year:

Table 2-2 Months of Year

Jan	Feb	Mar	Apr	May	Jun
Jul	Aug	Sep	Oct	Nov	Dec

`dd` specifies the day of the month.

`hh:mm:ss` specifies the time in hours, minutes, and seconds.

`yyyy` specifies the year.

`tp` is a pointer to the broken-down time structure.

Note

`asctime()` returns a pointer to a static area which may be overwritten. To insure data integrity, use the string or save it immediately.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`ctime()`
`localtime()`
`gmtime()`
`sys_mktime()`
`strftime()`

asin()

Arc Sine Function

Syntax

```
#include <math.h>
double asin(double x);
```

Description

`asin()` returns the arc sine of `x`, in the range [-π/2, +π/2] radians. A domain error, `EDOM` stored in `errno`, occurs for parameters not in the range `[-1, +1]`. On error, `asin()` returns `HUGE_VAL`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Error

`EDOM`

MICROWARE™

assert()

Place Diagnostics in Programs

Syntax

```
#include <assert.h>
void assert(int expression);
```

Description

`assert()` is a macro that places diagnostics in programs. If `expression` is false, `assert()` writes a message about the failed call to standard error. The following message is printed:

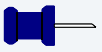
```
Assertion failed: expression, file filename,
     line line#\n
```

`expression` is the text of the parameter.

`filename` is the name of the file containing the `assert` macro.

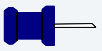
`line#` is the line number of the file containing the `assert` macro.

After printing the message, `assert()` calls `abort()`. This raises the `SIGABRT` signal that may be handled by the application.



Note

`assert()` never returns a value.



Note

If `NDEBUG` is defined as a macro name when the `assert.h` header file is included, `assert()` is defined as follows:

```
#define assert(ignore) ((void)0)
```

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`assert()` is a macro defined in `assert.h`.

See Also

`abort()`
`raise()`
`signal()`

MICROWARE™

atan()

Arc Tangent Function

Syntax

```
#include <math.h>
double atan(double x);
```

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Description

`atan()` returns the arc tangent of `x`, in the range [-π/2, +π/2] radians.

Library

`clib.l`

atan2()

Arc Tangent Function

Syntax

```
#include <math.h>
double atan2(
     double       y,
     double       x);
```

Description

`atan2()` returns the arc tangent of y divided by x (that is, `atan(y/x)`), in the range [-π, π] radians. `atan2()` uses the signs of both y and x to determine the quadrant of the returned value. A domain error, `EDOM` placed in `errno`, occurs if both y and x are zero. On error, `atan2()` returns `HUGE_VAL`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Error

`EDOM`

atexit()

Specify Function to Call at Normal Program Termination

Syntax

```
#include <stdlib.h>
int atexit(void (*func)(void));
```

Description

`atexit()` specifies a function to be called without parameters at normal program termination. Thirty-two functions may be specified.

`func` is a pointer to the function.

`atexit()` returns zero if it succeeds. Otherwise, it returns a non-zero value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`exit()`

atof()

Alpha to Floating Conversion

Syntax

```
#include <stdlib.h>
double atof(const char *string);
```

Description

`atof()` converts a string into its equivalent representation in type `double`.

`string` is a pointer to the string.

Except that no error indication is given, `atof()` is equivalent to the following:

```
strtod(nptr,(char **)NULL)
```

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`strtod()`

atoi()

Alpha to Integer Conversion

Syntax

```
#include <stdlib.h>
int atoi(const char *string);
```

Description

`atoi()` converts a string into its equivalent representation in type `int`.

`string` is a pointer to the string.

Except that no error indication is given, `atoi()` is equivalent to the following:

```
(int) strtol(nptr,(char**)NULL,10)
```

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`strtol()`

atol()

Alpha to Long Conversion

Syntax

```
#include <stdlib.h>
long atol(const char *string);
```

Description

`atol()` converts a string into its equivalent representation in type `long`.

`string` is a pointer to the string.

Except that no error indication is given, `atol()` is equivalent to the following:

```
strtol(nptr,(char**)NULL,10)
```

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`strtol()`

MICROWARE™

_aton()

Alpha to Numeric Translation

Syntax

```
#include <aton.h>
_number_type _aton(
     char          **buf,
     _number       *value);
```

Description

`_aton()` converts a string into its associated numeric value, if possible.

`buf` is a pointer to a pointer to a string containing a printable representation of a number expressed in base ten. The number may be an integer or a real number. You may also express the number in the following exponential format:

```
<number>(E|e)(-|+)<exponent>
```

`value` is a pointer to one of the three possible values that can be returned:

Table 2-3 _aton() Return Values

Value	Description
`_ERROR`	No recognizable number is found, or the value cannot be represented (it would cause overflow or underflow).
`_INTEGER`	The recognized number is an integer.
`_REAL`	The recognized number is not an integer.

`_aton()` updates `buf` to point to the character just after the recognized string. It updates `value` to contain the recognized number, unless an error occurs.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`clib.l`

_atou()

Alpha to Unsigned Conversion

Syntax

```
#include <stdlib.h>
unsigned _atou(char *string);
```

Description

`_atou()` converts a string into its appropriate unsigned numeric value, if possible.

`string` is a pointer to a string containing a printable representation of a number expressed in the format:

```
[+/-]<digits>
```

`_atou()` treats `long` and `int` values identically.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`clib.l`

attach()

Attach to Device

Syntax

```
#include <modes.h>
#include <io.h>
dev_list *attach(
     char         *name,
     int          mode);
```

Description

`attach()` causes a new device to become known to the system or verifies that the device is already attached.

If the device descriptor is found and the device is not already attached, `attach()` links to its file manager and device driver, and places their addresses in a new device table entry.

`name` is a pointer to the name of the device descriptor.

`mode` is the access mode:

```
FAM_READFAM_WRITES_IREADS_IWRITE
```

`mode` may be used to verify that subsequent read and/or write operations are permitted. Only the lower 16 bits of `mode` are used.

If the attach fails, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

_os_attach()	
I$Attach	***OS-9 for 68K Technical Manual***
I_ATTACH	***OS-9 Technical Manual***

bcmp()

Compares Bytes from Two Memory Areas

Syntax

```
#include <UNIX/os9def.h>
int bcmp(
     void         *addr1,
     void         *addr2,
     unsigned int len);
```

Description

`bcmp()` compares the memory at the first address (`addr1`) against the memory at the second address (`addr2`). The function returns zero if the memory areas are identical, non-zero otherwise. Both memory areas are assumed to be the given length (`len`). If `len` is zero, the function returns zero.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

See Also

`bcopy()`
`bzero()`
`ffs()`

bcopy()

Copies Bytes from One Memory Area to Another Memory Area

Syntax

```
#include <UNIX/os9def.h>
void  bcopy(
     void         *addr1,
     void         *addr2,
     unsigned int  len);
```

Description

`bcopy()` copies bytes, of length `len`, from one memory area (`addr1`) to another (`addr2`). Overlapping memory areas are handled correctly.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`unix.l`

See Also

`bcmp()`
`bzero()`
`ffs()`

bsearch()

Search Array

Syntax

```
#include <stdlib.h>
void *bsearch(
     const void   *key,
     const void   *base,
     size_t       nmemb,
     size_t       size,
     int (*compar)(const void *, const void *));
```

Description

`bsearch()` searches an array of `nmemb` objects for an element that matches `key`.

`key` is a pointer to the key object.

`base` is a pointer to the initial element of the array.

`nmemb` specifies the number of array elements.

`size` specifies the size of each array element.

`compar` is a pointer to a comparison function. `compar` is called with two parameters that point to the key object and to an array element, in that order. `compar` returns an integer that is:

- Negative if `key` is considered less than the array element.
- Zero if `key` is equal to the array element.
- Positive if `key` is considered greater than the array element.

The array consists of all elements that compare less than, equal to, and greater than the `key` object, in that order.

`bsearch()` returns a pointer to a matching array element. If no match is found, it returns a null pointer. If two elements compare as equal, which element is matched is unspecified.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

buildpath()

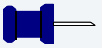
Adds a Filename to the Root Directory Variable

Syntax

```
#include <UNIX/os9def.h>
char  *buildpath(char *filename);
```

Description

`buildpath()` adds `filename` to the value of the environment variable `XOS9ROOTDIR`. If `XOS9ROOTDIR` is not set, a default value of `/h0` is used.



Note

`buildpath()` returns the final string in a static buffer that should be saved if additional calls are made to `buildpath()`. For example, with `XOS9ROOTDIR` set to `/dd/ETC`:

```
buildpath(xfile);
```

returns the string: `/dd/ETC/xfile`

without `XOS9ROOTDIR` set:

```
buildpath(xfile);
```

returns the string: `/h0/xfile`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

bzero()

Clear Bytes In Memory

Syntax

```
#include <UNIX/os9def.h>
void  bzero(
     void         *addr,
     unsigned int len);
```

Description

`bzero()` clears bytes in memory starting from an address (`addr`) for a number of bytes (`len`).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`unix.l`

See Also

`bcopy()`
`bcmp()`
`ffs()`

calloc()

Allocate Storage for Array

Syntax

```
#include <stdlib.h>
void *calloc(
     size_t       nmemb,
     size_t       size);
```

Description

`calloc()` allocates space for an array. The allocated memory is cleared to zeroes.

`nmemb` is the number of elements in the array.

`size` is the size of each array element.

If the allocation is successful, `calloc()` returns a pointer to the area. If the allocation fails or if `size` is zero, `NULL` is returned. The space designated by the returned pointer is suitably aligned to hold an object of any type.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

ebrk()
free()
malloc()
_os_srqmem()
_os9_srqmem()
realloc()

cbrt()

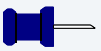
Cube Root

Syntax

```
#include <UNIX/os9def.h>
double cbrt(double x);
```

Description

`cbrt()` returns the cube root of `x`.



Note

The declaration `double cbrt()` must be included in any code that uses `cbrt()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

ceil()

Ceiling Function

Syntax

```
#include <math.h>
double ceil(double x);
```

Description

`ceil()` returns the smallest integer, as a `double`, that is not less than `x`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`floor()`

chainc(), chain()

Load and Execute New Module

Syntax

```
#include <process.h>
int chainc(
     char         *modname,
     int          parmsize,
     char         *parmptr,
     int          type,
     int          lang,
     int          datasize,
     int          prior,
     int          pathent);
int chain(
     char         *modname,
     int          parmsize,
     char         *parmptr,
     int          type,
     int          lang,
     int          datasize,
     int          prior);
```

Description

`chainc()` executes a new program without the overhead of creating a new process. It is functionally similar to `os9forkc()` followed by `exit()`, but with less system overhead.

`chainc()` effectively resets the calling process' program and data areas and begins executing a new primary module. Open paths are not closed or otherwise affected.

`modname` is a pointer to a null-terminated module name.

`parmsize` is generally `strlen(parmptr)`.

`parmptr` is a pointer to a null-terminated string to be passed to the new module.

`type` and `lang` specify the desired type and language of the module; values of zero indicate any type or language. Only the lower 16 bits of `type` and `lang` are used.

`datasize` allocates extra memory to the new program. If no extra memory is required, `datasize` can be zero.

`prior` is the new priority at which to run the program. Specify zero if no priority change is desired. Only the lower 16 bits of `prior` are used.

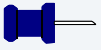
`pathent` is the number of open paths for the new process to inherit. Only the lower 16 bits of `pathent` are used.

Regardless of whether the chain is successful, `chainc()` never returns to the caller; you must verify that the program to chain to exists and is executable before chaining. Use `modlink()` to check the module directory for the program or `modload()` to check the execution directory.

`chain()` is the same as *c*`hainc()` without the `pathent` parameter. The number of open paths for the new process to inherit is three.

For More Information

`_os_exec()` is the preferred method by which to chain to C programs.



Note

Beware of chaining to a system state program. `chain()` is an historical function and is likely to be removed in a future release. `_os_exec()` is the recommended function.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`modlink()`
`modload()`
`_os_chain()`
`os9fork(), os9forkc()`
`_os_fork()`
`_os_exec()`
`F$Chain`, `F$Fork` — ***OS-9 for 68K Technical Manual***
`F_CHAIN`, `F_FORK` — ***OS-9 Technical Manual***

MICROWARE™

change_const()

Change Current Static Storage Pointer

Syntax

```
#include <regs.h>
void *change_const(void *constptr);
```

Description

`change_const()` changes the current constant data pointer and returns the previous constant data pointer.

`constptr` is the value of the new constant data pointer.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

change_static()

Change Current Static Storage Pointer

Syntax

```
#include <regs.h>
void *change_static(void *dest_stat);
```

Description

`change_static()` changes the current static storage pointer and returns the address of the previous static storage.

`dest_stat` is a pointer to the new static storage.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

chdir()

Change Current Data Directory

Syntax

```
#include <modes.h>
int chdir(char *dirname);
```

Description

`chdir()` changes the current data directory for the calling process.

`dirname` is a pointer to a string containing a path name for the directory.

`chdir()` returns zero after a successful call. If `dirname` is not a directory path name or some other error occurs, `-1` is returned; and the appropriate error code is placed in the global variable `errno`.

Note

`chdir()` changes the data directory only for the program containing the function call, not the shell that executes the program. Use the built-in shell command `chd` to change the shell's data directory.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_chdir()`	
`I$ChgDir`	***OS-9 for 68K Technical Manual***
`I_CHDIR`	***OS-9Technical Manual***
`chd`	***Using OS-9 for 68K*** and ***Using OS-9***

MICROWARE™

chmod()

Change File Access Permissions

Syntax

```
#include <modes.h>
int chmod(
     char         *name,
     int          perm);
```

Description

`chmod()` changes the access permission bits associated with a file.

`name` must be a pointer to a string containing a file name.

`perm` should contain the desired bit pattern for the file permissions. Only the lower 16 bits of `perm` are used.

`chmod()` returns zero after a successful call. If the caller is not entitled to change the access permissions or if the file cannot be found, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Note

Only the super user or the owner of the file may change the access permissions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ss_attr()`	
`I$SetStt`	***OS-9 for 68K Technical Manual***
`I_SETSTAT`, `SS_ATTR`	***OS-9 Technical Manual***
`attr`	***Using OS-9 for 68K*** and ***Using OS-9***

MICROWARE™

chown()

Change Owner of File

Syntax

```
#include <modes.h>
int chown(
     char          *name,
     int           newowner);
```

Description

`chown()` changes the owner number of a file.

`name` is a pointer to the name of the file.

`newowner` is the new owner ID to assign to the file. The owner ID consists of a group ID and a user ID.

For More Information

Refer to your operating system technical manual for information about how the owner ID is stored.

`chown()` returns zero after a successful call. If the caller does not have permission to change the owner ID or the file cannot be found, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Note

Only the super user may change the owner ID.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ss_fd()`	
`I$SetStt`	***OS-9 for 68K Technical Manual***
`I_SETSTAT, SS_FD`	***OS-9 Technical Manual***

chxdir()

Change Current Execution Directory

Syntax

```
#include <modes.h>
int chxdir(char *dirname);
```

Description

`chxdir()` changes the current execution directory for the calling process.

`dirname` is a pointer to a string containing the directory name.

`chxdir()` returns `0` after a successful call. If `dirname` is not a directory path name or some other error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Note

`chxdir()` changes the execution directory only for the program containing the function call, not the shell that executes the program. Use the built-in shell command `chx` to change the shell's execution directory.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_chdir()`	
`I$ChgDir`	***OS-9 Technical Manual***
`I_CHDIR`	***OS-9000 Technical Manual***
`chx`	***Using OS-9 for 68K*** and ***Using OS-9***

_cleareof(), cleareof()

Clear End of File Condition

Syntax

```
#include <stdio.h>
void _cleareof(FILE *stream);
void cleareof(FILE *stream);
```

Description

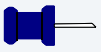
`_cleareof()` is a macro that resets the end-of-file condition that causes `feof()` to return a non-zero value.

`stream` is a pointer to the stream.

You can use this to retry an input operation on a terminal after end-of-file is encountered. `getc()` does not read characters until the end-of-file condition is cleared.

Note

If you are executing the compiler in strictly conforming ANSI mode, only `_cleareof()` is available. Otherwise, both calls are available.



Note

These calls are implemented as macros in the `stdio.h` header file.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

sys_clib.l

See Also

feof()
getc(), getchar()

clearerr()

Clear Error Condition

Syntax

```
#include <stdio.h>
void clearerr(FILE *stream);
```

Description

`clearerr()` clears the end-of-file and error indicators for a stream if compiled in strictly conforming ANSI mode. Otherwise, only the error indicator is cleared.

`stream` is a pointer to the stream to clear.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`_cleareof(), cleareof()`
`feof()`
`ferror()`
`fopen()`
`getc(), getchar()`

clock()

Get Processor Time

Syntax

```
#include <time.h>
clock_t  clock(void);
```

Description

`clock()` returns a value (in tick units) approximating the processor time used by the current process. The returned value may be divided by `CLOCKS_PER_SEC` to determine the processor time in seconds. `CLOCKS_PER_SEC` is defined in the `time.h` header file to be the number of ticks per second.

The era for `clock()` is the elapsed ticks for this process since the program was forked.

If `clock()` cannot determine the processor time, it returns `(clock_t) -1` to indicate an error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`F$SetSys` and `F$Time`	***OS-9 for 68K Technical Manual***
`F_SETSYS` and `F_TIME`	***OS-9 Technical Manual***

MICROWARE™

close()

Close Path

Syntax

```
#include <modes.h>
int close(int path);
```

Description

`close()` closes an open path. The path number is usually obtained by a previous call to `open()`, `creat()`, `create()`, or `dup()`. The standard paths 0, 1, and 2 (standard input, standard output, and standard error, respectively) are not normally closed by user programs.

`path` specifies the open path to close.

If an error occurs during the close, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Note

Be sure to use only a path number, not a file pointer assigned by `fopen()`.

Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

open()
creat()
create()
dup()
_os_close()

closedir()

Close Named Directory Stream

Syntax

```
#include <dir.h>
void closedir(DIR *dirp);
```

Description

`closedir()` closes the named directory stream and frees the structure associated with `dirp`.

`dirp` is a pointer to the directory stream.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

`opendir()`
`readdir()`
`rewinddir()`
`seekdir()`
`telldir()`

_cmpnam()

Compare Two Strings

Syntax

```
#include <strings.h>
error code _cmpnam(
     uchar        *pattern,
     uchar        *string,
     uint32       length);
```

Description

`_cmpnam()` compares the target string to the pattern string to determine if they match.

`target` is a pointer to the target string. The target name must be terminated by a null byte. Upper and lower case characters are considered to match.

`pattern` is a pointer to the pattern string. Two metacharacters are recognized in the pattern string:

- A question mark (?) matches any single character.
- An asterisk (*) matches any string of characters.

`patlen` specifies the length of the pattern string.

`_cmpnam()` returns zero if the strings match. `-1` is returned if no match occurs.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_cmpnam()`	
F$CmpNam	***OS-9 for 68K Technical Manual***
F_CMPNAM	***OS-9 Technical Manual***

cos()

Cosine Function

Syntax

```
#include <math.h>
double cos(double x);
```

Description

`cos()` returns the cosine of `x` as a double. The value of `x` is in radians. A domain error, `EDOM` placed in `errno`, occurs if `x` is ∞ or -∞. `cos()` returns `HUGE_VAL` on error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Error

`EDOM`

MICROWARE™

cosh()

Hyperbolic Cosine Function

Syntax

```
#include <math.h>
double cosh(double x);
```

Description

`cosh()` returns the hyperbolic cosine of `x`. The value of `x` is in radians. If the magnitude of `x` is too large, a range error, `ERANGE` placed in `errno`, occurs and the properly signed `HUGE_VAL` is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Error

`ERANGE`

_cpymem()

Copy External Memory

Syntax

```
#include <process.h>
int _cpymem(
     int          pid,
     int          count,
     void         *from,
     void         *into);
```

Description

`_cpymem()` copies memory owned by another process or the system into a buffer.

`pid` is the process ID number of the external process. If `pid` is zero, the system's address space is assumed. Only the lower 16 bits of `pid` are used.

`count` is the number of bytes to copy.

`from` is a pointer to the address in the process' address space from which to copy.

`into` is a pointer to the buffer into which to copy the memory.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_cpymem()`	
F$CpyMem	***OS-9 for 68K Technical Manual***
F_CPYMEM	***OS-9 Technical Manual***

crc()

Calculate Module CRC

Syntax

```
#include <module.h>
int crc(
     void         *ptr,
     unsigned int count,
     int          *accum);
```

Description

`crc()` generates or checks the CRC (cyclic redundancy check) values of sections of memory. Compilers, assemblers, and other module generators use `crc()` to generate a valid module CRC.

`ptr` is a pointer to the data.

`count` specifies the byte count for the data.

The CRC accumulator is passed at the location pointed to by `accum`. `crc()` also stores the updated CRC accumulator at the location pointed to by `accum`.

To generate the CRC for a module:

1. Initialize the accumulator to `-1 (0xffffffff)`.
2. Perform the CRC over the module.
3. Call `crc()` with a NULL value for `ptr`.
4. Complement the CRC accumulator.
5. Write the contents of the accumulator to the module.

For More Information

Refer to your operating system technical manual for more information on how your operating system uses CRCs for modules.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_crc()`

F$CRC	***OS-9 for 68K Technical Manual***
F_CRC	***OS-9 Technical Manual***

Example

The following code fragment illustrates a technique for generating a CRC for a module:

```
#include <stdio.h>
#include <module.h>

main()
{
  char *ptr;
  int count,accum = -1;
  /* one or more calls to crc */
  crc(ptr,count,&accum);

  /* The final call to crc(). At */
  crc(ptr,count,&accum);

/* this point the entire module less the four crc
/* bytes has been processed */
  crc(NULL,0,&accum);
    /* this is a special case to   */
    /* run a null byte through the */
    /* crc  calculation */

  accum = ~accum; /* complement the accumulator.*/
                  /* Previous calls have left the   */
                  /*  most significant byte as 0xff */
```

```
                    /* Complementing changes this byte*/
                    /* to zero */

  fwrite(&accum,1,sizeof accum,fp);
    /* finally write out the four crc bytes  */
}
```

creat()

Create File

Syntax

```
#include <modes.h>
int creat(
     char         *name,
     int          mode);
```

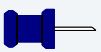
Description

`creat()` returns a path number to a newly created file. The file is available for writing.

`name` is a pointer to the newly created file. If `name` is the name of an existing file, the file is truncated to zero length and the ownership and permissions remain unchanged.

`mode` specifies the access modes for the file. The `modes.h` header file contains the valid `mode` values. Only the lower 16 bits of `mode` are used.

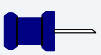
If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

You do not need to specify `write` access in `mode` to write to the file.

You cannot create directories with this call; instead, use `_os_makdir()`.



Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`open()`
`_os_create()`
`_os_makdir()`
`_os_ss_size()`

`I$Create` and `I$SetStt`	***OS-9 for 68K Technical Manual***
`I_CREATE` and `I_SETSTAT`	***OS-9 Technical Manual***
`SS_SIZE`	***OS-9 Technical Manual***

MICROWARE™

create()

Create File

Syntax

```
#include <modes.h>
int create(
     char         *name,
     int          mode,
     int          perm,
     {int initial_size});
```

Description

`create()` returns a path number to the newly created file.

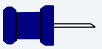
`name` is a pointer to the newly created file.

`mode` specifies the access modes for the file. The `modes.h` header file contains the valid `mode` values. Only the lower 16 bits of `mode` are used.

`perm` specifies the file permission attributes. The `modes.h` header file contains the valid `perm` values. Only the lower 16 bits of `perm` are used.

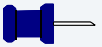
`initial_size` may be used to indicate the file's initial allocation size. For disk files, the allocation is made even though the file size does not change. For pipe files, the pipe buffer is set to this value. `initial_size` may be used only if the `FAM_SIZE` (or `S_ISIZE`) bit is set in `mode`.

If the file already exists or any other error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

`create()` is similar to the `creat()` call. However, `create()` allows the caller to specify the exact file attributes desired and does not truncate the file if it is already present. You cannot create directories with this call; instead, use `_os_makdir()`.



Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_create()`
`_os_makdir()`
`_os_ss_size()`

`I$Create` and `I$SetStat`	***OS-9 for 68K Technical Manual***
`I_CREATE` and `I_SETSTAT`	***OS-9 Technical Manual***
`SS_SIZE`	***OS-9Technical Manual***

crtolf()

Convert Carriage Return Characters to Linefeed Characters

Syntax

```
#include <UNIX/os9def.h>
void crtolf(
     char         *buf,
     unsigned int n);
```

Description

`crtolf()` converts all carriage return characters to linefeed characters in `buf` for a span of n characters.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

ctime()

Convert Calendar Time to String Format

Syntax

```
#include <time.h>
char *ctime(time_t *t);
```

Description

`ctime()` converts the calendar time `t` into the following 26-byte string format, including the terminating `\0`:

```
xxx mmm dd hh:mm:ss yyyy\n\0
```

`xxx` is one of the following days of the week:

```
Sun  Mon  Tue  Wed  Thu  Fri  Sat
```

`mmm` is one of the following months of the year:

```
Jan  Feb  Mar  Apr  May  Jun
Jul  Aug  Sep  Oct  Nov  Dec
```

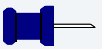
`dd` specifies the day of the month.

`hh:mm:ss` specifies the time in hours, minutes, and seconds.

`yyyy` specifies the year.

`ctime(t)` is the equivalent of:

```
asctime(localtime(t))
```



Note

`ctime()` returns a pointer to a static area which may be overwritten. To ensure data integrity, use the string or save it immediately.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`asctime()`
`gmtime()`
`localtime()`

detach()

Detach Device

Syntax

```
#include <io.h>
#include <modes.h>
int detach(dev_list *ptr);
```

Description

`detach()` removes a device from the system device table if no other process is using it.

`ptr` is a pointer returned by either `attach()` or `_os_attach()`.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_attach()`
`_os_detach()`
`I$Detach` — ***OS-9 for 68K Technical Manual***
`I_DETACH` — ***OS-9 Technical Manual***

MICROWARE™

difftime()

Find Temporal Difference

Syntax

```
#include <time.h>
double difftime(
     time_t       time1,
     time_t       time0);
```

Description

`difftime()` returns the difference in seconds between `time1` and `time0`. This value is returned as a `double`. If you could subtract the `time_t` values, the return value would be: `time1 - time0`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`time()`

div()

Compute Quotient and Remainder

Syntax

```
#include <stdlib.h>
div_t div(
     int          numer,
     int          denom);
```

Description

`div()` returns the quotient and remainder of the division of the numerator by the denominator.

`numer` is the numerator.

`denom` is the denominator.

If the division is inexact, the resulting quotient is the integer of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is undefined. Otherwise, `quot * denom + rem` equals `numer`.

`div()` returns a structure of type `div_t`, comprising both the quotient and the remainder. The `div_t` structure contains the `quot` and `rem` fields.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

MICROWARE™

_dtoa()

Double to ASCII Conversion

Syntax

```
#include <stdlib.h>
int _dtoa(
     double       fnum,
     char         *buff,
     int          tot_digs,
     int          frac_digs,
     int          *sign);
```

Description

`_dtoa()` converts a floating point number of type `double` to the ASCII string equivalent. The conversion terminates when:

- The requested number of digits are converted.
- The specified number of digits after the decimal point is reached.

`fnum` is the floating point number to convert.

`buff` is a pointer to a buffer for the converted ASCII string. `_dtoa()` returns the signed decimal exponent. The returned string only contains mantissa digits. A null byte is appended to the end of `buff`. `buff` should be one byte larger than the requested number of digits.

`tot_digs` is the number of digits requested.

`frac_digs` is the specified number of digits following the decimal point.

`sign` is a pointer to the sign. Set the sign in `sign` as follows:

- `1` for a negative sign.
- `0` for a positive sign.

If `fnum` represents ±∞ or an IEEE 754 Not a Number and `tot_digs` is greater than or equal to three, `buff` contains, respectively, `Inf` or `NaN`. If `tot_digs` is less than three, `buff` contains, respectively, `I` or `N`. In these cases, `_dtoa()` returns `DBL_MAX_10_EXP`.

`_dtoa()` returns no errors.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`clib.l`

MICROWARE™

dup()

Duplicate Path

Syntax

```
#include <modes.h>

int dup(int path);
```

Description

`dup()` returns a synonymous path number for an existing file or device. The lowest available path number is used.

`path` specifies the path number.

`dup()` increments the link count of a path descriptor and returns a different synonymous path number. The path descriptor is not cloned.

Note

It is usually not a good idea for more than one process to perform I/O on the same path concurrently.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_dup()`

`I$Dup`	***OS-9 for 68K Technical Manual***
`I_DUP`	***OS-9 Technical Manual***

dup2()

Specifies the Value of Duplicated Path

Syntax

```
#include <UNIX/os9def.h>
int dup2(int fd1, int fd2);
```

Description

`dup2()` returns a synonymous path number for an existing file or device and specifies the value of the new duplicated path.

`fd1` specifies the original path number.

`fd2` specifies the value of the duplicated path. If path `fd2` is already in use, it is first closed.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

ebrk()

Get External Memory

Syntax

```
#include <stdlib.h>
extern int _memmins;
void *ebrk(unsigned int size);
```

Description

`ebrk()` returns the amount of memory specified by `size`. The memory is obtained from the system using the `F$SRqMem` (for OS-9 for 68K) or `_os_srqmem()` (for OS-9) system request. It is intended for general purpose memory allocation.

To reduce the overhead involved in requesting small quantities of memory, `ebrk()` requests memory from the system in a minimum size determined by the global variable `_memmins` and satisfies the user requests from this memory space. `_memmins` is initially set to 8192. `ebrk()` grants memory requests from this memory space provided the requests are no larger than the amount of space.

If the request is larger than the available space, `ebrk()` wastes the rest of the space and tries to get enough memory from the system to satisfy the request.

This method works well for programs that need to get large amounts of not necessarily contiguous memory in little bits and cannot afford the overhead of `malloc()`.

Changing the `_memmins` variable causes `ebrk()` to use that value as the `F$SRqMem` (for OS-9 for 68K) or `F_SRQMEM` (for OS-9) memory request size.

If the memory request is granted, a pointer (even-byte aligned) to the block is returned. If the request is not granted, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Note

The memory obtained from `ebrk()` is not returned until the process terminates.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

`sbrk()`
`ibrk()`
`malloc()`
`_os_srqmem()`

`F$SRqMem`	***OS-9 for 68K Technical Manual***
`F_SRQMEM`	***OS-9 Technical Manual***

endpwent()

Closes Password File

Syntax

```
#include <UNIX/pwd.h>
void endpwent();
```

Description

`endpwent()` can be used to close a password file after processing is complete.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

See Also

`fgetpwent()`
`getpw()`
`fgetpwent()`
`getpwnam()`
`getpwuid()`
`setpwent()`

_errmsg()

Print Error Message

Syntax

```
#include <stdio.h>
int _errmsg(
     int          nerr,
     char         *msg,
     [u_int32     arg1,
     u_int32      arg2,
     u_int32      arg3]);
```

Description

`_errmsg()` displays an error message and the program name on the standard error path.

`nerr` specifies the error number returned by `_errmsg()`.

`msg` is a pointer to the error message. The message string is displayed in the following format:

```
prog: <message text>
```

`prog` is the module name of the program, and `<message text>` is the string pointed to by `msg`.

For added flexibility in message printing, `msg` can be a conversion string suitable for `fprintf()` with up to three additional parameters of any integral type. `nerr` is returned as the value of the function so `_errmsg()` can be used as a parameter to a function such as `exit()` or `prerr()`.

Example

Assume the program calling the function is named `foobar`:

```
Call:     errmsg(1,"programmed message\n");
Prints:   toobar: programmed message
Call:     exit(_errmsg(errno,
               "unknown option '%c'\n",'q'));
Prints:   foobar: unknown option 'q'
```

Then exits with `errno` as its termination status.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

`fprintf()`
`_prgname()`

Syntax

```
#include <events.h>
int _ev_creat(
     int          ev_value,
     int          wait_inc,
     int          signal_inc,
     char         *ev_name);
```

Description

`_ev_creat()` creates an event.

`ev_value` is the initial value for the event.

`wait_inc` is the wait increment. It is added to the value of the event when a successful `_ev_wait()`, `_ev_waitr()`, `_os_ev_wait()`, or `_os_ev_waitr()` is performed.

`signal_inc` is the signal increment. It is added to the value of the event when an `_ev_signal()` or `_os_ev_signal()` is performed.

`ev_name` is a pointer to the name for the event.

An event ID number is returned if the event is successfully created.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_ev_signal()`	
`_ev_wait()`	
`_ev_waitr()`	
`_os_ev_creat()`	
`_os_ev_signal()`	
`_os9_ev_wait()`	
`_os_ev_waitr()`	
`F$Event`	***OS-9 for 68K Technical Manual***
`F_EVENT`	***OS-9 Technical Manual***
`EV_CREAT`	***OS-9 Technical Manual***
`EV_SIGNL`	***OS-9 Technical Manual***
`EV_WAIT`	***OS-9 Technical Manual***
`EV_WAITR`	***OS-9 Technical Manual***

_ev_delete()

Delete Event

Syntax

```
#include <events.h>
int _ev_delete(char *ev_name);
```

Description

`_ev_delete()` deletes an event. The use count for the event must be zero before the event can be deleted.

`ev_name` is a pointer to the name of the event.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ev_delete()`

`F$Event`	***OS-9 for 68K Technical Manual***
`F_EVENT`, `EV_DELET`	***OS-9 Technical Manual***

_ev_info()

Get Event Information

Syntax

```
#include <events.h>
int _ev_info(
     int          ev_index,
     int          ev_infostr,
     int          *ev_buffer);
```

Description

`_ev_info()` returns information about an event. The event table is indexed from zero to one less than the maximum number of events allowed on the system.

`ev_index` corresponds to the starting point in the event table to search for an event.

`ev_buffer` is a pointer to the event structure buffer used to hold the event information, if found.

If `ev_index` is greater than all active events in the table, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

_os_ev_info()
F$Event
F_EVENT, EV_INFO

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_ev_link()

Link to Existing Event

Syntax

```
#include <events.h>
int _ev_link(char *ev_name);
```

Description

`_ev_link()` links to an existing event. An event ID number is returned if the event is successfully linked.

`ev_name` is a pointer to the name of the event.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ev_link()`

`F$Event`	***OS-9 for 68K Technical Manual***
`F_EVENT, EV_LINK`	***OS-9 Technical Manual***

Syntax

```
#include <events.h>
int _ev_pulse(
     int          ev_id,
     int          ev_value,
     int          allflag);
```

Description

`_ev_pulse()` indicates that an event has occurred. The current event value is saved. The event variable is set to the value given by `ev_value`, and the normal signal increment is not applied. The saved event value is restored after activating processes, if any.

`ev_id` is the event ID returned from a call to `_ev_creat()`, `_ev_link()`, `_os_ev_creat()`, or `_os_ev_link()`.

`ev_value` sets the value of the event variable.

`allflag` indicates which process(es) to activate. Only the lower 16 bits of `allflag` are used.

- If `allflag` is zero, the first process waiting for the event is activated.
- If `allflag` is `0x8000`, all processes waiting for the event that have a value in range are activated.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`. Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

_ev_creat()
_ev_link()
_os_ev_creat()
_os_ev_link()
_os_ev_pulse()

F$Event	***OS-9 for 68K Technical Manual***
F_EVENT	***OS-9 Technical Manual***
V_CREAT	***OS-9 Technical Manual***
EV_LINK	***OS-9 Technical Manual***
EV_PULSE	***OS-9 Technical Manual***

_ev_read()

Read Event Without Waiting

Syntax

```
#include <events.h>
int _ev_read(int ev_id);
```

Description

`_ev_read()` reads the value of an event, without waiting for or affecting the event variable, and returns the current value of the event.

`ev_id` identifies the event.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`. Clear the global variable `errno` prior to calling `_ev_read()` to distinguish between return of an event value of `-1` and an error.

For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

Example

```
errno = 0;
if ((ev_val = _ev_read(ev_id)) == -1 && errno)
    exit(_errmsg(errno,
     "can’t read event value - "));
```

See Also

_os_ev_read()

F$Event	***OS-9 for 68K Technical Manual***
F_EVENT	***OS-9 Technical Manual***
EV_READ	***OS-9 Technical Manual***

Syntax

```
#include <events.h>
int _ev_set(
     int          ev_id,
     int          ev_value,
     int          allflag);
```

Description

`_ev_set()` indicates that an event has occurred. The event variable is set to the value given by `ev_value`, and the normal signal increment is not applied. Processes waiting for the event are then activated.

`ev_id` is the event ID returned from a call to `_ev_creat()`, `_ev_link()`, `_os_ev_creat()`, or `_os_ev_link()`.

`ev_value` sets the value of the event variable.

`allflag` specifies the process(es) to activate. Only the lower 16 bits of `allflag` are used.

- If `allflag` is zero, the first process waiting for the event is activated.
- If `allflag` is `0x8000`, all processes waiting for the event that have a value in range are activated.

OS-9 for 68K: `_ev_set()` returns the value of the event before changing to `ev_value`. If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`. Clear the global variable `errno` prior to calling `_ev_set()` to distinguish between return of a prior event value of `-1` and an error.

OS-9: `_ev_set()` returns zero. If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_ev_creat()`	
`_ev_link()`	
`_os_ev_creat()`	
`_os_ev_link()`	
`_os_ev_set()`	
`F$Event`	***OS-9 for 68K Technical Manual***
`F_EVENT`	***OS-9 Technical Manual***
`EV_CREAT`	***OS-9 Technical Manual***
`EV_LINK`	***OS-9 Technical Manual***
`EV_SET`	***OS-9 Technical Manual***

Syntax

```
#include <events.h>
int _ev_setr(
     int          ev_id,
     int          ev_value,
     int          allflag);
```

Description

`_ev_setr()` indicates that an event has occurred. The event variable is incremented by the value given by `ev_value`. Processes waiting for the event are then activated.

`ev_id` is the event ID returned from a call to `_ev_creat()`, `_ev_link()`, `_os_ev_creat()`, or `_os_ev_link()`.

`ev_value` specifies the amount to increment the event variable.

`allflag` specifies the process(es) to activate. Only the lower 16 bits of `allflag` are used.

- If `allflag` is zero, the first process waiting for the event is activated.
- If `allflag` is `0x8000`, all processes waiting for the event that have a value in range are activated.

OS-9 for 68K: `_ev_setr()` returns the value of the event before changing by `ev_value`. If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`. Clear the global variable `errno` prior to calling `_ev_setr()` to distinguish between return of a prior event value of `-1` and an error.

OS-9: `_ev_setr()` returns `0`. If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_ev_creat()`	
`_ev_link()`	
`_os_ev_creat()`	
`_os_ev_link()`	
`_os_ev_setr()`	
`F$Event`	***OS-9 for 68K Technical Manual***
`F_EVENT`	***OS-9 Technical Manual***
`EV_CREAT`	***OS-9 Technical Manual***
`EV_LINK`	***OS-9 Technical Manual***
`EV_SETR`	***OS-9 Technical Manual***

Syntax

```
#include <events.h>
int _ev_signal(
     int          ev_id,
     int          allflag);
```

Description

`_ev_signal()` indicates that an event has occurred. The current event variable is updated by the signal increment which was given when the event was created.

`ev_id` is the event ID returned from a call to `_ev_creat()`, `_ev_link()`, `_os_ev_creat()`, or `_os_ev_link()`.

`allflag` specifies the process(es) to activate. Only the lower 16 bits of `allflag` are used.

- If `allflag` is zero, the first process waiting for the event is activated.
- If `allflag` is `0x8000`, all processes waiting for the event that have a value in range are activated.

OS-9 for 68K: `_ev_signal()` returns the value of the event before incremented by the signal increment. If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`. Clear the global variable `errno` prior to calling `_ev_signal()` to distinguish between return of a prior event value of `-1` and an error.

OS-9: `_ev_signal()` returns zero. If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_ev_creat()`	
`_ev_link()`	
`_os_ev_creat()`	
`_os_ev_link()`	
`_os_ev_signal()`	
`F$Event`	***OS-9 for 68K Technical Manual***
`F_EVENT`	***OS-9 Technical Manual***
`EV_CREAT`	***OS-9 Technical Manual***
`EV_LINK`	***OS-9 Technical Manual***
`EV_SIGNL`	***OS-9 Technical Manual***

_ev_unlink()

Unlink Event

Syntax

```
#include <events.h>
int _ev_unlink(int ev_id);
```

Description

`_ev_unlink()` decrements the link count of an event. When the link count becomes zero, you can delete the event using `_ev_delete()` or `_os_ev_delete()`.

`ev_id` specifies the event ID.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_ev_delete()`
`_os_delete()`
`_os_ev_unlink()`

F$Event	***OS-9 for 68K Technical Manual***
F_EVENT	***OS-9 Technical Manual***
EV_DELET	***OS-9 Technical Manual***
EV_UNLNK	***OS-9 Technical Manual***

Syntax

```
#include <events.h>
int _ev_wait(
     int          ev_id,
     int          ev_min,
     int          ev_max);
```

Description

`_ev_wait()` waits for an event to occur. The event value is compared to the range values specified by `ev_min` and `ev_max`. If the event value is not in the specified range, the process waits until some other process places the value within the range. Once in range, the wait increment is applied to the event value. The actual event value is returned as the value of the function.

`ev_id` specifies the event ID.

`ev_min` specifies the minimum range value for the event.

`ev_max` specifies the maximum range value for the event.

`_ev_wait()` returns the value of the event after completion of the wait. If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`. Clear the global variable `errno` prior to calling `_ev_wait()` to distinguish between return of a current event value of `-1` and an error. Further, if a signal was received by the process, the event value returned is outside of the range `ev_min` to `ev_max`.

```
errno = 0;
if ((ev_val = _ev_wait(ev_id, -5, 0)) == -1 && errno)
     exit(_errmsg(errno,
          "error waiting on event - "));
if (ev_val < -5 || ev_val > 0)
    _errmsg(1,
    "received signal while waiting for event\n");
```

For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ev_wait()`	
`_os9_ev_wait()`	
`F$Event`	***OS-9 for 68K Technical Manual***
`F_EVENT`	***OS-9 Technical Manual***
`EV_WAIT`	***OS-9 Technical Manual***

Syntax

```
#include <events.h>
int _ev_waitr(
     int          ev_id,
     int          ev_min,
     int          ev_max);
```

Description

`_ev_waitr()` waits for the specified event to occur. The event value is compared to the relative values specified by `ev_min` and `ev_max`. The current event value is added to the range values before the comparison. If the event value is not in the specified range, the process waits until some other process places the value within the range. Once in range, the wait increment is applied to the event value. The actual event value is returned as the value of the function.

`ev_id` specifies the event ID for the event.

`ev_min` specifies the minimum range value for the event.

`ev_max` specifies the maximum range value for the event.

`_ev_waitr()` returns the value of the event after completion of the wait. If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`. Clear the global variable `errno` prior to calling `_ev_waitr()` to distinguish between return of a current event value of `-1` and an error. Further, if a signal was received by the process, the event value returned is outside of the relative range `ev_min` to `ev_max`.

For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ev_waitr()`	
`F$Event`	***OS-9 for 68K Technical Manual***
`F_EVENT`, `EV_WAITR`	***OS-9 Technical Manual***

execl()

Calls a File With Variable Arguments

Syntax

```
#include <UNIX/os9def.h>
int execl(
     char         *path,
     char         *arg0,
     ...);
```

Description

`execl()` is the UNIX-like exec interface to the `_os_chain` system call.

`path` is the pathlist to the program to chain to.

`arg0`, and any additional variable arguments, are the arguments to pass to the chained process. `arg0` is conventionally the name of the chained program. The list of variable arguments is terminated by a NULL entry.

The global variable `_environ` is used for the environment for the call to `_os_chain()`.

`execl()` returns `-1` and sets the global variable `errno` to indicate an error. `execl()` only returns an error if setting up for the `_os_chain()` call fails. Once an attempt to chain occurs, the process exits if any error occurs.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

See Also

execle()
execv()
execvp()

execle()

Calls a File With Variable Arguments

Syntax

```
#include <UNIX/os9def.h>
int execle(
     char         *path,
     char         *arg0,
     ...,
     char         *envp[]);
```

Description

`execle()` is the UNIX-like exec interface to the `_os_chain` system call.

`path` is the pathlist to the program to chain to.

`arg0`, and any additional variable arguments, are the arguments to pass to the chained process. `arg0` is conventionally the name of the chained program. The list of variable arguments is terminated by a `NULL` entry. The `NULL` entry is followed by an environment pointer to use for the call to `_os_chain()`. `envp` is a pointer to an array of strings that constitute the environment of the process.

`execle()` returns `-1` and sets the global variable `errno` to indicate an error. `execle()` only returns an error if setting up for the `_os_chain()` call fails. Once an attempt to chain occurs, the process exits if any error occurs.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

See Also

execl()
execv()
execvp()

execv()

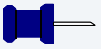
Calls a File With an Argument Vector

Syntax

```
#include <UNIX/os9def.h>
int execv(
     char         *path,
     char         *argv[]);
```

Description

`execv()` is a UNIX-like interface to the `_os_chain` system call. The argument to `execv()` is an argument vector to pass to the chained process and the path to the program to chain to. The last argument should be a `NULL` entry. `argv[0]` is conventionally the name of the program being chained to.



Note

To use `execv()`, link with `unix.l`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

See Also

`execl()`
`execle()`
`execvp()`

execve()

Calls a File With an Argument Vector

Syntax

```
#include <UNIX/os9def.h>
int execv(
     char         *path,
     char         *argv[],
     char         *envp[]);
```

Description

`execve()` is a UNIX-like interface to the `_os_chain` system call. The argument to `execve()` is an argument vector to pass to the chained process and the path to the program to chain to. The last argument should be a `NULL` entry. `argv[0]` is conventionally the name of the program being chained to. The argument `envp` is a pointer to a null-terminated array of character pointers to null-terminated strings.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

See Also

`execl()`
`execle()`
`execvp()`

execvp()

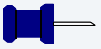
Calls a File With an Argument Vector

Syntax

```
#include <UNIX/os9def.h>
int execvp(
     char         *file,
     char         *argv[]);
```

Description

`execvp()` is a UNIX-like interface to the `_os_chain` system call. The argument to *e*`xecvp()` is an argument vector to pass to the chained process and the path to the program to chain to. The last argument should be a `NULL` entry. `argv[0]` is conventionally the name of the program being chained to.



Note

`_os_chain()` does not do the normal PATH searching that the `shell` does.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

See Also

`execl()`
`execle()`
`execv()`

Syntax

```
#include <stdlib.h>
void _exit(int status);
```

Description

`_exit()` immediately terminates a program.

`status` is the exit status. The parent process can use `status` to determine the program's success or failure.

An exit `status` of zero is considered normal termination. Most programs (especially the `shell`) interpret a non-zero value as an error code.

`_exit()` cannot return to its caller, and it does not flush and close standard I/O buffers.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_exit()`	
`F$Exit`	***OS-9 for 68K Technical Manual***
`F_EXIT`	***OS-9 Technical Manual***

Syntax

```
#include <stdlib.h>
void exit(int status);
```

Description

`exit()` is the normal means of terminating a task. All functions specified by `atexit()` are called in the reverse order of the order in which they were specified.

All open streams with unwritten buffered data are flushed, all open streams are closed, and all files created by `tmpfile()` are removed. Control is returned to the host environment.

`status` is the exit status.

- If `status` is zero or `EXIT_SUCCESS`, zero is communicated to the parent process executing the `_os_wait()`.
- If `status` is `EXIT_FAILURE`, one is communicated.

`exit()` cannot return to its caller.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`atexit()`	
`_os_exit()`	
`_os_wait()`	
`tmpfile()`	
F$Exit	***OS-9 for 68K Technical Manual***
F_EXIT	***OS-9 Technical Manual***

exp()

Exponential Function

Syntax

```
#include <math.h>
double exp(double x);
```

Description

`exp()` returns the exponential function of `x`, `e` (2.71828..) raised to the `x` power. A range error, `ERANGE` stored in `errno`, occurs if the magnitude of `x` is too large. In this case, `exp()` returns `HUGE_VAL`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Error

`ERANGE`

fabs()

Floating Absolute Value

Syntax

```
#include <math.h>
double fabs(double x);
```

Description

`fabs()` returns the absolute value of `x`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

fclose()

Close File

Syntax

```
#include <stdio.h>
int fclose(FILE *stream);
```

Description

`fclose()` flushes the file pointed to by `stream` and closes the associated file. Any unwritten buffered data for the stream are written to the file; any unread buffered data are discarded. The stream is disassociated from the file. If the associated buffer was automatically allocated, it is deallocated. If `stream` was returned from `tmpfile()`, it is deleted.

`stream` is a pointer to the C I/O `FILE` structure.

`fclose()` returns zero if the stream was successfully closed. Otherwise, `EOF` is returned.

For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `fclose()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

fflush()
fopen()
getc(), getchar()
putc(), putchar()
tmpfile()

fdopen()

Attach Path to File Pointer

Syntax

```
#include <stdio.h>
FILE *fdopen(
     int          path,
     char         *action);
```

Description

`fdopen()` returns a file pointer to the file specified by a currently open path.

`path` is the path to attach.

`action` is a pointer to the file action string. The `action` must be compatible with the access mode of the given path. The valid actions are listed in **Table 2-4**.

Table 2-4 Valid Actions

Action	Description
`r`	Open for reading
`w`	Open for writing
`a`	Append (write) at the end of the file or create file for writing if **path** does not exist
`r+`	Open for update
`w+`	Create for update
`a+`	Create or open for update at end of file
`d`	Directory read

Use `fdopen()` when you require some special processing not provided by `fopen()` for opening files.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

`fopen()`
`freopen()`

feof()

Check Buffered File for End of File

Syntax

```
#include <stdio.h>
int feof(FILE *stream);
```

Description

`feof()` tests the end-of-file indicator for the specified stream. `feof()` returns a non-zero value only if the end-of-file indicator is set for `stream`.

`stream` is a pointer to the C I/O `FILE` structure.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`clearerr()`
`fopen()`

ferror()

Check Buffered File for Error Condition

Syntax

```
#include <stdio.h>
int ferror(FILE *stream);
```

Description

`ferror()` tests the error indicator for the specified stream. `ferror()` returns a non-zero value only if the error indicator is set for `stream`.

`stream` is a pointer to the C I/O `FILE` structure.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`clearerr()`
`fopen()`

fflush()

Flush File's Buffer

Syntax

```
#include <stdio.h>
int fflush(FILE *stream);
```

Description

`fflush()` writes any unwritten data for the specified stream to the file. Unwritten data is written if the stream was opened in either `write` or `update` mode.

`stream` is a pointer to the C I/O `FILE` structure. If `stream` is a null pointer, `fflush()` performs this flushing action on all streams opened in either `write` or `update` mode and containing unwritten data.

`fflush()` returns `EOF` if a write error occurs or if the stream is read-only. Otherwise, it returns zero.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fopen()`
`getc(), getchar()`
`putc(), putchar()`

ffs()

Find First Set Bit

Syntax

```
#include <UNIX/os9def.h>
int ffs(unsigned int i);
```

Description

`ffs()` finds the first bit set in the argument passed to it and returns the index of that bit. Bits are numbered starting at one (1) from the least significant. A return value of zero indicates that the value passed to the function was zero.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

fgetc()

Get Character from File

Syntax

```
#include <stdio.h>
int fgetc(FILE *stream);
```

Description

`fgetc()` returns a character, as an unsigned converted to an `int`, from the file and advances the associated file position indicator for the stream, if defined.

`stream` is a pointer to the C I/O `FILE` structure.

If the stream is at end-of-file, the end-of-file indicator for the stream is set and `fgetc()` returns `EOF`. If a read error occurs, the error indicator for the stream is set and `fgetc()` returns `EOF.`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`getc(), getchar()`

fgetpos()

Get Current Position in File

Syntax

```
#include <stdio.h>
int fgetpos(
     FILE         *stream,
     fpos_t       *pos);
```

Description

`fgetpos()` stores the current value of the file position indicator. The stored value contains information usable by `fsetpos()` for repositioning the stream to its position at the time of the `fgetpos()` call.

`stream` is a pointer to the C I/O `FILE` structure.

`pos` is a pointer to the file position indicator.

If successful, `fgetpos()` returns zero. Otherwise, it returns a non-zero value and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fsetpos()`

fgetpwent()

Get Password File Entry

Syntax

```
#include <UNIX/pwd.h>
struct passwd *fgetpwent(FILE *f);
```

Description

`fgetpwent()` returns a pointer to the next password structure in the stream (`f`), which matches the format of the password file `/dd/SYS/password`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

See Also

`endpwent()`
`getpw()`
`getpwent()`
`getpwnam()`
`getpwuid()`
`setpwent()`

fgets()

Get String from File

Syntax

```
#include <stdio.h>
char *fgets(
     char         *ptr,
     int          cnt,
     FILE         *stream);
```

Description

`fgets()` reads at most one less than the specified number of characters from the stream into an array. No additional characters are read after a newline character (which is returned) or after end-of-file. A null character is added to the end of the string.

`ptr` is a pointer to the array into which to read the characters.

`cnt` is the number of characters to read.

`stream` is a pointer to the C I/O `FILE` structure.

`fgets()` returns `ptr` if successful. If the end-of-file is encountered and no characters have been read into the array, the contents of the array remain unchanged and a null pointer is returned. If a read error occurs during the operation, the array contents are indeterminate and a null pointer is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`clib.l`

See Also

`fgetc()`

_fileno(), fileno()

Determine Path Number from File

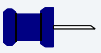
Syntax

```
#include <stdio.h>
int _fileno(FILE *stream);
int fileno(FILE *stream);
```

Description

`_fileno()` returns the path number associated with the specified file pointer. The path number is not valid unless `(stream->_flag & _INIT)` is non-zero. This call is only available as a macro.

`stream` is a pointer to the C I/O `FILE` structure.



Note

If you are executing the compiler in strictly conforming ANSI mode, only `_fileno()` is available. Otherwise, both calls are available.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

These macros are defined in the `stdio.h` header file.

See Also

`fopen()`
`freopen()`

findnstr()

Search String for Pattern

Syntax

```
#include <strings.h>
int findnstr(
     int          pos,
     char         *string,
     char         *pattern,
     int          len);
```

Description

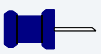
`findnstr()` searches the string for the first occurrence of the pattern. It starts at position `pos` (where the first position is one, not zero). The returned value is the position of the first matched character of the pattern in the string, or zero if a match is not found. `findnstr()` stops searching at `len`; it may continue past null bytes.

`pos` specifies the starting position.

`string` is a pointer to the string to search.

`pattern` is a pointer to the pattern to use for the search.

`len` specifies the final position to be searched.



Note

The current implementation does not use the most efficient pattern matching algorithm. `strstr()` is preferred for very long strings.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library
sys_clib.l

See Also
findstr()
strstr()

findstr()

Search String for Pattern

Syntax

```
#include <strings.h>
int findstr(
     int          pos,
     char         *string,
     char         *pattern);
```

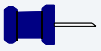
Description

`findstr()` searches the string for the first instance of the pattern. It starts at position `pos` (where the first position is one, not zero). The returned value is the position of the first matched character of the pattern in the string, or zero if a match is not found. `findstr()` stops searching when a null byte is found in `string`.

`pos` specifies the starting position.

`string` is a pointer to the string to search.

`pattern` is a pointer to the pattern to use for the search.



Note

The current implementation does not use the most efficient pattern matching algorithm. `strstr()` is preferred for very long strings.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`findnstr()`
`strstr()`

floor()

Floor Function

Syntax

```
#include <math.h>
double floor(double x);
```

Description

`floor()` returns the largest integer, as a `double`, that is not greater than `x`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`ceil()`

fmod()

Compute Floating Point Remainder

Syntax

```
#include <math.h>
double fmod(
     double       x,
     double       y);
```

Description

`fmod()` computes the floating-point remainder of `x` divided by `y`. If `y` is non-zero, `fmod()` returns the value with the same sign as `x` and a magnitude less than the magnitude of `y`. If `y` is zero, zero is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Error

`EDOM`

fopen()

Open File

Syntax

```
#include <stdio.h>
FILE *fopen(
     const char   *name,
     const char   *mode);
```

Description

`fopen()` opens a file and associates a stream with it. If an error occurs, `fopen()` returns NULL.

`name` is a pointer to the name of the file.

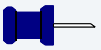
`mode` is a pointer to the access mode. `mode` may be one of the following:

Table 2-5 fopen() Mode types

Mode	Description
`r`	Open text file for reading
`w`	Truncate to zero length or create text file for writing
`a`	Append; open or create text file for writing at end-of-file
`rb`	Open binary file for reading
`wb`	Truncate to zero length or create binary file for writing
`ab`	Append; open or create binary file for writing at end-of-file
`r+`	Open text file for update (reading and writing)

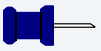
Table 2-5 fopen() Mode types (continued)

Mode	Description
w+	Truncate to zero length or create text file for update
a+	Append; open or create text file for update, writing at end-of-file
r+b, rb+	Open binary file for update (reading and writing)
w+b, wb+	Truncate to zero length or create binary file for update
a+b, ab+	Append; open or create binary file for update, writing at end-of-file
d	Directory read (OS-9 extension); this is only available in K & R source mode.



Note

Appending an x to any action(s) indicates that names are relative to the current execution directory. The x also implies that the file should have the execute permission bit set. This option is only available for OS-9 in non-ANSI mode.



Note

In modes other than compatibility mode, appending a d to any action(s) indicates that the file is a directory.

`mode` must point to a string and not a character; `fopen("fun","r")` is correct, `fopen("fun",'r')` is not correct.

If the file does not exist or cannot be read, opening a file with read mode fails.

When you open a file with append mode, the file position indicator is initially positioned at the beginning of the file. However, all subsequent writes to the file are forced to the then current end-of-file, regardless of intervening calls to `fseek()`. If an error occurs when opening a file with append mode, refer to **fopen Append Bit** subsection in the appropriate processor-specific chapter of the ***Ultra C/C++ Processor Guide***.

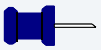
When a file is opened with update mode, both input and output may be performed on the associated stream. However, output may not be directly followed by input without an intervening call to `fflush()` or to a file positioning function (`fseek()`, `fsetpos()`, or `rewind()`). Likewise, input may not be directly followed by output without an intervening call to a file positioning function, unless the input operation encounters end-of-file.

When opened, a stream is fully buffered only if it does not refer to an interactive device. The error and end-of-file indicators for the stream are cleared.

Three file pointers are available and are considered open the moment the program runs:

`stdin`	standard input	I/O is to path 0
`stdout`	standard output	I/O is to path 1
`stderr`	standard error	I/O is to path 2

These macros are defined in the `stdio.h` header file.



Note

You cannot open a directory for writing with `fopen()`.

Example

Examples using `fopen()` are found in **Table 2-6**.

Table 2-6 fopen() Examples

`fp = fopen(fred,wx);`	Creates a file for writing in the execution directory
`fp = fopen(moe,r);`	Opens an existing file for reading
`fp = fopen(stooge/curly,w+);`	Creates a file for reading and writing

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`creat()`
`fflush()`
`fseek()`
`getc(), getchar()`
`open()`
`_os_create()`
`_os9_create()`
`_os_open()`
`putc(), putchar()`

Syntax

```
#include <stdio.h>
int fprintf(
     FILE         *stream,
     const char   *format, ...);
```

Description

`printf()`, `fprintf()`, and `sprintf()` are standard C library functions that perform formatted output. Each of these functions converts, formats, and prints any parameters as indicated by the control string.

`fprintf()` writes output to a stream.

`stream` is a pointer to the C I/O `FILE` structure.

`format` is a pointer to a control string. `format` determines the format, type, and number of parameters that the function expects. If the control string does not correctly match the parameters, the results are not predictable.

`fprintf()` returns when the end of the format string is encountered.

A format specification consists of a percent `(%)` character followed by (in this order):

1. Zero or more flags that modify the meaning of the conversion specification. You may use the following flags in any order:

Table 2-7 fprintf() Flags

Flag	Description
`-`	The result of the conversion is left-justified within the field. If this flag is not specified, the result is right-justified.
`+`	The result of a signed conversion always begins with a plus or minus sign. If this flag is not specified, the result begins with a sign only when a negative value is converted.
`space`	If the first character of a signed conversion is not a sign or if a signed conversion results in no characters, a space is prefixed to the result. If the space and + flags both appear, the space flag is ignored.
`#`	The result is to be converted to an alternate form. For `o` conversion, it increases the precision to force the first digit of the result to zero. For `x` (or `X`) conversion, a non-zero result has a `0x` (or `0X`) prefix. For `e`, `E`, `f`, `g`, and `G` conversions, the result always contains a decimal point character, even if no digits follow. Normally, a decimal point character appears in the result of these conversions only if a digit follows it. For `g` and `G` conversions, trailing zeros are not removed from the result. For other conversions, the behavior is undefined.
`0`	For `d`, `i`, `o`, `u`, `x`, `X`, `e`, `E`, `f`, `g`, and `G` conversions, leading zeros, following any indication of sign or base, pad the field width; no space padding is performed. If the `0` and `-` flags both appear, the `0` flag is ignored. For `d`, `i`, `o`, `u`, `x`, and `X` conversions, the `0` flag is ignored if a precision is specified. For other conversions, the behavior is undefined.

2. An optional minimum field width. If the converted value has fewer characters than the field width, it is padded with spaces (by default) on the left (or right, if the left adjustment flag has been given) to the field width. The field width takes the form of an asterisk (*) or a decimal integer.

3. An optional precision that gives:

- The minimum number of digits to appear for the d, i, o, u, x, and X conversions.
- The number of digits to appear after the decimal point character for e, E, and f conversions.
- The maximum number of significant digits for the g and G conversions.
- The maximum number of characters to be written from a string in s conversion.

  The precision takes the form of a period (.) followed either by an asterisk (*) or by an optional decimal integer. If only the period is specified, the precision is taken as zero. If a precision appears with any other conversion specifier, the behavior is undefined.

4. An optional h, l, or L that specifies:

Table 2-8 fprintf() Optional flag descriptions

Flag	Description
h	The following d, i, o, u, x, or X conversion specifier applies to a short int or an unsigned short int parameter, or the following n conversion specifier applies to a pointer to a short int parameter.
l	The following d, i, o, u, x, or X conversion specifier applies to a long int or an unsigned long int parameter, or the following n conversion specifier applies to a pointer to a long int parameter.
L	The following e, E, f, g, or G conversion specifier applies to a long double parameter.

If an h, l, or L appears with any other conversion specifier, the behavior is undefined.

5. A character that specifies the type of conversion to apply.

   An asterisk (`*`) may indicate the field width, precision, or both. In this case, an `int` parameter supplies the field width or precision. The parameters specifying field width, precision, or both appear in that order before the parameter, if any, to convert.

   A negative field width parameter is taken as a `-` flag followed by a positive field width.

   A negative precision parameter is taken as if the precision was omitted.

The conversion specifiers and their meanings are:

Table 2-9 fprintf() Conversion Specifiers

Flag	Description
`c`	The `int` parameter is converted to an `unsigned char`, and the resulting character is written.
`d, i`	The `int` parameter is converted to a signed decimal in the style `[-]dddd`. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is `1`. The result of converting a zero value with a precision of zero is no characters.
`e, E`	The `double` parameter is converted in the style `[-]d.ddde±dd`, where there is one digit before the decimal point character (which is non-zero if the parameter is non-zero) and the number of digits after the decimal point character is equal to the precision. The default precision is `6`. If the precision is zero and the `#` flag is not specified, no decimal point character appears. The value is rounded to the appropriate number of digits. The `E` conversion specifier produces a number with `E` instead of `e` introducing the exponent. The exponent always contains at least two digits. If the value is zero, the exponent is zero.

Table 2-9 fprintf() Conversion Specifiers (continued)

Flag	Description
f	The double parameter is converted to decimal notation in the style [-]ddd.ddd. The number of digits after the decimal point character is equal to the precision specification. The default precision is 6. If the precision is zero and the # flag is not specified, no decimal point character appears. If a decimal point character appears, at least one digit appears before it. The value is rounded to the appropriate number of digits.
g, G	The double parameter is converted in the style f or e (or in style E in the case of a G conversion specifier), with the precision specifying the number of significant digits. If the precision is zero, it is taken as one. The style used depends on the value converted; style e (or E) is used only if the exponent resulting from the conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional portion of the result. A decimal point character appears only if it is followed by a digit.
n	The parameter is a pointer to an integer into which is written the number of characters written to the output stream so far by this call to fprintf(). No parameter is converted.
o, u, x, X	The unsigned int parameter is converted to unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal notation (x or X) in the style dddd. The letters abcdef are used for x conversion and letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is one. The result of converting a zero value with a precision of zero is no characters.
p	The parameter is a pointer to a void. The pointer value is converted to a sequence of printable characters as an unsigned hexadecimal integer.

Table 2-9 fprintf() Conversion Specifiers (continued)

Flag	Description
s	The parameter is a pointer to an array of character type. Characters from the array are written up to, but not including, a terminating null character. If the precision is specified, no more than that many characters are written. If the precision is not specified or is greater than the size of the array, the array contains a null character.
%	A % is written. No parameter is converted. The complete conversion specification is %%.

If a conversion specification is invalid, the behavior is undefined.

If any parameter is, or points to, a union or an aggregate (except for an array of character type using %s conversion or a pointer using %p conversion), the behavior is undefined.

A non-existent or small field width never truncates a field. If the result of a conversion is wider than the field width, the field expands to contain the conversion result.

`fprintf()` returns the number of characters transmitted, or a negative value if an output error occurred.

For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `fprintf()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

printf()
sprintf()

Syntax

```
#include <stdio.h>
int fputc(
     int          chr,
     FILE         *stream);
```

Description

`fputc()` converts a character to an `unsigned char`. It writes the character to the output stream at the position indicated by the associated file position indicator for the stream, if defined, and advances the indicator appropriately. If the file cannot support positioning requests or if the stream was opened with append mode, the character is appended to the output stream.

`chr` is the character to convert.

`stream` is a pointer to the C I/O `FILE` structure.

`fputc()` returns the character written. If a write error occurs, the error indicator for the stream is set and `fputc()` returns `EOF`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fgetc()`

fputs()

Output String to File

Syntax

```
#include <stdio.h>
int fputs(
     const char   *str,
     FILE         *stream);
```

Description

`fputs()` writes a string to a stream. The terminating null character is not written.

`str` is a pointer to the string.

`stream` is a pointer to the stream to write.

`fputs()` returns `EOF` if a write error occurs. Otherwise, it returns a non-negative value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fgets()`

fread()

Read Data from File

Syntax

```
#include <stdio.h>
int fread(void *ptr, size_t size, size_t nmemb,
          FILE *stream);
```

Description

`fread()` reads up to `nmemb` elements into an array. The file position indicator for the stream, if defined, is advanced by the number of characters successfully read. If an error occurs, the resulting value of the file position indicator for the stream is indeterminate. If a partial element is read, its value is indeterminate.

`ptr` is a pointer into the array.

`size` specifies the size of each array element.

`nmemb` specifies the number of array elements to read.

`stream` is a pointer to the C I/O `FILE` structure.

`fread()` returns the number of elements successfully read, which may be less than `nmemb` if a read error or end-of-file is encountered. If `size` or `nmemb` is zero, `fread()` returns zero and the array contents and stream state remain unchanged.

For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `fread()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fwrite()`

Syntax

```
#include <stdlib.h>
void free(void *ptr);
```

Description

`free()` deallocates the space pointed to by `ptr`. If `ptr` is a null pointer, no action occurs. Otherwise, if the parameter does not match a pointer previously returned by `calloc()`, `malloc()`, or `realloc()`, or if the space has been deallocated by a call to `free()` or `realloc()`, the behavior is undefined.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	no
Standards:	ANSI

Library

`clib.l`

See Also

`calloc()`
`malloc()`
`realloc()`

_freemem(), freemem()

Determine Size of Unused Stack Area

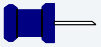
Syntax

```
#include <stdlib.h>
int freemem(void);
int _freemem(void);
```

Description

`freemem()` returns the number of bytes allocated for the stack that have not been used.

If compiler stack checking is enabled, the stack is checked for possible overflow before entering a function. The lowest address the stack pointer has reached is retained so `freemem()` can report the number of bytes between the stack limit and the lowest stack value as the unused stack memory.



Note

The program must be compiled with stack checking code in effect for `freemem()` to return a correct result. `freemem()` is historical; avoid using it in new code as it is likely to be removed in a future release. These calls are not available when compiled in strictly conforming ANSI mode.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`cstart.r`

See Also

`stacksiz(), _stacksiz()`

_freemin()

Set Memory Reclamation Bound

Syntax

```
#include <stdlib.h>
void _freemin(int size);
```

Description

`_freemin()` allows you to set the minimum size for a free block of memory or concatenation of free blocks for `free()` to return to the operating system. If a program is known to use a lot of memory, you can instruct `free()` never to return memory to the operating system.

`size` is the minimum size for a free block of memory or the concatenation of free blocks.

- If `size` is positive, it is used as the minimum number of bytes for a free block of memory to be a candidate for return to the operating system.
- If `size` is negative, `free()` never tries to return memory allocated by the ANSI memory allocation functions (`calloc()`, `malloc()`, or `realloc()`) to the system.

You do not need to use `_freemin()` to return memory to the system. The default minimum size to return is 4K. `_freemin()` ignores any specified non-negative `size` smaller than 4K.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	no
Standards:	ANSI

Library

`clib.l`

See Also

`calloc()`
`free()`
`malloc()`
`realloc()`

freopen()

Re-Open File

Syntax

```
#include <stdio.h>
FILE *freopen(
     char         *name,
     char         *action,
     FILE         *stream);
```

Description

`freopen()` substitutes a file name for the file currently open, if any. The original file is closed, even if the opening of the new file does not succeed.

`freopen()` is usually used to associate the `stdin`, `stdout`, or `stderr` file pointers with a file instead of a terminal device.

`name` is a pointer to the file name to use for a substitute.

`action` specifies the action to take. Refer to `fopen()` for details on the action values.

`stream` is a pointer to the C I/O `FILE` structure.

`freopen()` returns the `stream` passed or a null pointer if the open fails.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fopen()`

frexp()

Return Portions of Floating Point Number

Syntax

```
#include <math.h>

double frexp(
     double       value,
     int          *exp);
```

Description

`frexp()` breaks a floating point value into a normalized fraction and an integral exponent of two. `frexp()` returns the fraction `x`, such that `1/2 <= x < 1` and `value = x * 2^exp`.

`value` is the floating point value.

`exp` is a pointer to the integral exponent of two.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <stdio.h>
int fscanf(
     FILE         *stream,
     const char   *format, ...);
```

Description

`fscanf()` reads input from a stream.

`stream` is a pointer to the C I/O `FILE` structure.

`format` is a pointer to a character sequence. `format` specifies the allowable input sequences and specifies how to convert these sequences for assignments. If the format is exhausted while parameters remain, the excess parameters are evaluated, but are otherwise ignored.

The format is a multibyte character sequence, beginning and ending in its initial shift state. The format is composed of zero or more directives:

- One or more white space characters.
- An ordinary mutibyte character (neither `%` nor a white space character).
- A conversion specification.

A percent (`%`) character introduces each conversion specification. After the `%`, the following appear in sequence:

- An optional assignment-suppressing asterisk (`*`) character.
- An optional non-zero decimal integer that specifies the maximum field width.
- An optional `h`, `l`, or `L` indicating the size of the receiving object.

  The conversion specifiers `d`, `i`, and `n` are preceded by `h` if the corresponding parameter is a pointer to a `short int` rather than pointing to `int`, or by `l` if it is a pointer to a `long int`.

The conversion specifiers o, u, and x are preceded by h if the corresponding parameter is a pointer to an `unsigned short int` rather than pointing to an `unsigned int`, or by `l` if it is a pointer to an `unsigned long int`.

The conversion specifiers e, f, and g are preceded by `l` if the corresponding parameter is a pointer to a `double` rather than pointing to a `float`, or by `L` if it is a pointer to a `long double`.

If an h, `l`, or `L` appears with any other conversion specifier, the behavior is undefined.

- A character that specifies the type of conversion to apply. The following conversion specifiers are valid:

Table 2-10 fscanf() Conversion Specifiers

Flag	Description
c	Matches a character sequence of the number specified by the field width. By default, the field width is one. The corresponding parameter is a pointer to the initial character of an array large enough to accept the sequence. No null character is added.
d	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of `strtol()`. The base parameter is 10. The corresponding parameter is a pointer to an integer.
e, f, g	Matches an optionally signed floating point number, whose format is the same as expected for the subject string of `strtod()`. The corresponding parameter is a pointer to a float.
i	Matches an optionally signed integer, whose format is the same as expected for the subject sequence of `strtol()`. The base parameter is `0`. The corresponding parameter is a pointer to an integer.
n	No input is consumed. The corresponding parameter is a pointer to an integer into which to write the number of characters read from the input stream so far by this call to `fscanf()`. Executing a %n directive does not increment the assignment count returned when `fscanf()` finishes execution.

Table 2-10 fscanf() Conversion Specifiers (continued)

Flag	Description
o	Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of `strtoul()`. The base parameter is 8. The corresponding parameter is a pointer to an unsigned integer.
p	Matches the set of sequences that may be produced by the %p conversion of `fprintf()`. The corresponding parameter is a pointer to a void. The input item is interpreted the same as %x. If the input item is a value converted earlier during the same program execution, the resulting pointer compares equal to that value. Otherwise, the behavior of the %p conversion is undefined.
s	Matches a sequence of non-white space characters. The corresponding parameter is a pointer to the initial character of an array large enough to accept the sequence and a terminating null character. The terminating null character is added automatically.
u	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of `strtoul()`. The base parameter is 10. The corresponding parameter is a pointer to an unsigned integer.
x	Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of `strtoul()`. The base parameter is 16. The corresponding parameter is a pointer to an unsigned integer.

Table 2-10 fscanf() Conversion Specifiers (continued)

Flag	Description
[	Matches a non-empty character sequence from a set of expected characters called the `scanset`. The corresponding parameter is a pointer to the initial character of an array large enough to accept the sequence and a terminating null character. The terminating null character is added automatically. The conversion specifier includes all subsequent characters in the format string, up to and including the matching right bracket (`]`). The characters between the brackets are the `scanlist`. The `scanlist` is the `scanset`, unless the character after the left bracket is a caret (`^`). In this case, the `scanset` contains all characters not appearing in the `scanlist` between the `^` and the `]`. If the conversion specifier begins with `[]` or `[^]`, the `]` is in the `scanlist` and the next right bracket character ends the specification. Otherwise, the first right bracket character ends the specification. A hyphen (`-`) has no special meaning; it is matched like any other character.
%	Matches a single %. No conversion or assignment occurs. The complete conversion specification is %%.

The conversion specifiers `E`, `G`, and `X` are also valid and behave the same as, respectively, `e`, `g`, and `x`.

If a conversion specification is invalid, the behavior is undefined.

`fscanf()` executes each directive of the format in turn. If a directive fails, `fscanf()` returns. Failures are described as **input failures** or **matching failures**.

- An input failure is caused by the unavailability of input characters.
- A matching failure is caused by inappropriate input.

A directive composed of white space character(s) is executed by reading input up to the first non-white space character or until no more characters can be read. The first non-white space character remains unread.

A directive containing an ordinary multibyte character is executed by reading the next characters of the stream. If one character differs from one in the directive, the directive fails, and the differing and subsequent characters remain unread.

A directive containing a conversion specification defines a set of matching input sequences for each specifier. A conversion specification is executed in the following steps:

- Input white space characters (as specified by `isspace()`) are skipped, unless the specification includes a `[`, `c`, or `n` specifier.
- An input item is read from the stream, unless the specification includes an `n` specifier. An input item is defined as the longest matching sequence of input characters, unless it exceeds a specified field width. If it exceeds the field width, it is the initial subsequence of that length in the sequence. The first character, if any, after the input item remains unread. If the length of the input item is zero, the execution of the directive fails. This condition is a matching failure, unless an error prevented input from the stream. If an error prevented input from the stream, it is an input failure.
- Except in the case of a `%` specifier, the input item (or, in the case of a `%n` directive, the count of input characters) is converted to a type appropriate to the conversion specifier. If the input item is not a matching sequence, the execution of the directive fails. This condition is a matching failure. Unless assignment suppression was indicated by an asterisk (`*`), the conversion result is placed in the object pointed to by the first parameter following the format parameter that has not already received a conversion result. If this object does not have an appropriate type or if the conversion result cannot be represented in the space provided, the behavior is undefined.

If the end-of-file is reached during input, conversion terminates. If end-of-file occurs before any characters matching the current directive have been read, other than leading white space where permitted, execution of the current directive terminates with an input failure. Otherwise, unless execution of the current directive is terminated with a matching failure, execution of the following directive, if any, is terminated with an input failure.

If conversion terminates on a conflicting input character, the offending input character is left unread in the input stream. Trailing white space, including newline characters, is left unread unless matched by a directive. You must use the %n directive to determine the success of literal matches and suppressed assignments.

`fscanf()` returns EOF if an input failure occurs before any conversion. Otherwise, `fscanf()` returns the number of input items assigned which can be fewer than provided for, or zero in the event of an early matching failure.

For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `fscanf()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`scanf()`
`sscanf()`
`strtod()`
`strtol()`
`strtoul()`

Syntax

```
#include <stdio.h>
int fseek(
     FILE         *stream,
     long int     offset,
     int          place);
```

Description

`fseek()` sets the file position indicator for a stream.

`stream` is a pointer to the C I/O `FILE` structure.

`offset` specifies an amount to offset the file position indicator.

`place` specifies where to place the file position indicator. `place` has three possible values:

Table 2-11 fseek() place Values

Value	Description
`SEEK_SET`	The new file position is offset.
`SEEK_CUR`	The new file position is the current file position + `offset`. `offset` can be negative.
`SEEK_END`	The new file position is the current file size + `offset` (usually `offset` is negative or zero in this case). A successful call to `fseek()` clears the end-of-file indicator for the stream and undoes the effects of the `ungetc()` function on the same stream. After an `fseek()` call, the next operation may be either input or output.

`fseek()` returns non-zero only for a request that cannot be satisfied.

For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `fseek()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`ftell()`
`getc(), getchar()`
`putc(), putchar()`
`ungetc()`

fsetpos()

Set Current File Position

Syntax

```
#include <stdio.h>
int fsetpos(
     FILE         *stream,
     const fpos_t *pos);
```

Description

`fsetpos()` sets the file position indicator for a stream.

`stream` is a pointer to the C I/O `FILE` structure.

`pos` is a pointer to the value of the file position indicator. `pos` is returned from `fgetpos()`.

A successful call to `fsetpos()` clears the end-of-file indicator for the stream and undoes any effects of `ungetc()` on the same stream. After an `fsetpos()` call, the next operation on an update stream may be either input or output.

If successful, `fsetpos()` returns zero. Otherwise, it returns a non-zero value and places the appropriate error code in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fgetpos()`

fstat()

Get File Status

Syntax

```
#include <UNIX/stat.h>
int fstat(
     unsigned int fd,
     struct stat  *buf);
```

Description

`fstat()` obtains the same information about an open file referenced by the argument descriptor, as would be obtained by a `stat` call.

`buf` is a pointer to a `stat` structure into which information about the file is placed.

`fstat()` returns zero on success. It returns `-1` on failure and sets `errno` to indicate the error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

See Also

`stat()`

ftell()

Report File Pointer Position

Syntax

```
#include <stdio.h>
long ftell(FILE *stream);
```

Description

`ftell()` returns the current value of the file position indicator as the number of bytes from the beginning of the file for a stream. This is not necessarily a meaningful measure of the number of characters written or read.

`stream` is a pointer to the C I/O `FILE` structure.

If successful, `ftell()` returns the current value of the file position indicator for the stream. On failure, `ftell()` returns `-1L` and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fseek()`
`getc(), getchar()`
`putc(), putchar()`

fwrite()

Write Data to File

Syntax

```
#include <stdio.h>
size_t fwrite(
     const void   *ptr,
     size_t       size,
     size_t       nmemb,
     FILE         *stream);
```

Description

`fwrite()` writes up to `nmemb` elements from an array to the stream. The file position indicator for the stream, if defined, is advanced by the number of characters successfully written. If an error occurs, the resulting value of the file position indicator for the stream is indeterminate.

`ptr` is a pointer to the array.

`size` specifies the size of each array element.

`nmemb` specifies the number of array elements to write.

`stream` is a pointer to the C I/O `FILE` structure.

`fwrite()` returns the number of elements successfully written. This is less than `nmemb` only if a write error is encountered.

A write to a text stream does not cause the associated file to truncate beyond that point.

For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `fwrite()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fread()`

get<name>()

Read Value of SPR, DCR, PREG, and TB

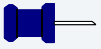
Syntax

```
#include <getset.h>
#include <types.h>
u_int32 _get_<name>(void);
```

Description

The `_get_<name>()` class of functions can be used to read the value of Special Purpose Registers (SPRs), Device Control Registers (DCRs), Processor Registers (PREGs), and Time-Based Registers (TBs).

`name` is any of the names listed in the following table. There is one exception to the following table, the `dec` SPR on the 601 can be read from user-state with the function `_get_user_dec()`.



Note

All functions are provided. It is the programmer's responsibility to avoid using system-state only registers from user-state and accessing registers that do not exist on the target processor. All registers are accessible from system-state. Only those noted are available from user-state. Attempting to use system state registers from user state causes unpredictable results.

Table 2-12 get<name> Valid Names and Associated Processors

<name>	Register Type	Processor(s) 403	505	601	603	User	Writable
bar	SPR		•				•
bear	DCR	•					
besr	DCR	•					•
br0	DCR	•					•
br1	DCR	•					•
br2	DCR	•					•
br3	DCR	•					•
br4	DCR	•					•
br5	DCR	•					•
br6	DCR	•					•
br7	DCR	•					•
cmpa	SPR		•				•
cmpb	SPR		•				•
cmpc	SPR		•				•
cmpd	SPR		•				•

Table 2-12 get<name> Valid Names and Associated Processors (continued)

<name>	Register Type	Processor(s) 403	505	601	603	User	Writable
cmpe	SPR		•				•
cmpf	SPR		•				•
cmpg	SPR		•				•
cmph	SPR		•				•
counta	SPR		•				•
countb	SPR		•				•
cr	PREG	•	•	•	•	•	•
ctr	SPR	•	•	•	•	•	•
dabr	SPR			•			•
dac1	SPR	•					•
dac2	SPR	•					•
dar	SPR		•	•	•		•
dbat0l	SPR				•		•
dbat0u	SPR				•		•
dbat1l	SPR				•		•
dbat1u	SPR				•		•

Table 2-12 get<name> Valid Names and Associated Processors (continued)

<name>	Register Type	Processor(s) 403	505	601	603	User	Writable
dbat2l	SPR				•		•
dbat2u	SPR				•		•
dbat3l	SPR				•		•
dbat3u	SPR				•		•
dbcr	SPR	•					•
dbsr	SPR	•					•
dccr	SPR	•					•
dcmp	SPR				•		•
dear	SPR	•					
dec	SPR		•	•	•		•
der	SPR		•				•
dmacc0	DCR	•					•
dmacr0	DCR	•					•
dmacr1	DCR	•					•
dmacr2	DCR	•					•
dmacr3	DCR	•					•

Table 2-12 get<name> Valid Names and Associated Processors (continued)

<name>	Register Type	Processor(s) 403	505	601	603	User	Writable
dmact0	DCR	•					•
dmact1	DCR	•					•
dmact2	DCR	•					•
dmact3	DCR	•					•
dmada0	DCR	•					•
dmada1	DCR	•					•
dmada2	DCR	•					•
dmada3	DCR	•					•
dmasa0	DCR	•					•
dmasa1	DCR	•					•
dmasa2	DCR	•					•
dmasa3	DCR	•					•
dmasr	DCR	•					•
dmiss	SPR				•		•
dpdr	SPR		•				•
dsisr	SPR		•	•	•		•

Table 2-12 get<name> Valid Names and Associated Processors (continued)

<name>	Register Type	Processor(s) 403	505	601	603	User	Writable
ear	SPR			•	•		•
ecr	SPR		•				•
eid	SPR		•				•
eie	SPR		•				•
esr	SPR	•					•
evpr	SPR	•					•
exier	DCR	•					•
exisr	DCR	•					•
fpecr	SPR		•				•
fpscr	PREG	•	•	•	•	•	•
hash1	SPR				•		•
hash2	SPR				•		•
hid0	SPR			•	•		•
hid1	SPR			•			•
hid15	SPR			•			•
hid2	SPR			•			•

Table 2-12 get<name> Valid Names and Associated Processors (continued)

<name>	Register Type	Processor(s) 403	505	601	603	User	Writable
hid5	SPR			•			•
iabr	SPR			•	•		•
iac1	SPR	•					•
iac2	SPR	•					•
ibat0l	SPR			•	•		•
ibat0u	SPR			•	•		•
ibat1l	SPR			•	•		•
ibat1u	SPR			•	•		•
ibat2l	SPR			•	•		•
ibat2u	SPR			•	•		•
ibat3l	SPR			•	•		•
ibat3u	SPR			•	•		•
icadr	SPR		•				•
iccr	SPR	•					•
iccst	SPR		•				•
icdat	SPR		•				•

Table 2-12 get<name> Valid Names and Associated Processors (continued)

<name>	Register Type	Processor(s) 403	505	601	603	User	Writable
icmp	SPR				•		•
ictrl	SPR		•				•
imiss	SPR				•		•
iocr	DCR	•					•
lctrl1	SPR		•				•
lctrl2	SPR		•				•
lr	SPR	•	•	•	•	•	•
mq	SPR			•		•	•
nri	SPR		•				•
msr	PREG	•	•	•	•		•
pbl1	SPR	•					•
pbl2	SPR	•					•
pbu1	SPR	•					•
pbu2	SPR	•					•
pir	SPR			•			•
pit	SPR	•					•

Table 2-12 get<name> Valid Names and Associated Processors (continued)

<name>	Register Type	Processor(s) 403	505	601	603	User	Writable
pvr	SPR	•		•	•		
rpa	SPR				•		•
rtcl	SPR			•			•
rtcu	SPR			•			•
sdr1	SPR			•	•		•
sprg0	SPR	•		•	•		•
sprg1	SPR	•		•	•		•
sprg2	SPR	•		•	•		•
sprg3	SPR	•		•	•		•
sr0	PREG			•	•		•
sr1	PREG			•	•		•
sr2	PREG			•	•		•
sr3	PREG			•	•		•
sr4	PREG			•	•		•
sr5	PREG			•	•		•
sr6	PREG			•	•		•

Table 2-12 get<name> Valid Names and Associated Processors (continued)

<name>	Register Type	Processor(s) 403	505	601	603	User	Writable	
sr7	PREG			•	•		•	
sr8	PREG			•	•		•	
sr9	PREG			•	•		•	
sr10	PREG			•	•		•	
sr11	PREG			•	•		•	
sr12	PREG			•	•		•	
sr13	PREG			•	•		•	
sr14	PREG			•	•		•	
sr15	PREG			•	•		•	
srr0	SPR	•	•	•	•		•	
srr1	SPR	•	•	•	•		•	
srr2	SPR	•					•	
srr3	SPR	•					•	
tbhi	SPR	•					•	
tbl	SPR		•		•		•	
tbl	TB		•		•	•	•	*

Table 2-12 get<name> Valid Names and Associated Processors (continued)

<name>	Register Type	Processor(s) 403	505	601	603	User	Writable	
tblo	SPR	•					•	
tbu	SPR		•		•		•	
tbu	TB		•		•	•	•	*
tcr	SPR	•					•	
tsr	SPR	•					•	
xer	SPR	•	•	•	•	•	•	

Legend:
User-state read version is `_get_user_<name>`
* Write version is system state only

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

See Also

`_set_<name>()`

getc(), getchar()

Get Next Character from File, Stdin

Syntax

```
#include <stdio.h>
int getc(FILE *stream);
int getchar(void);
```

Description

`getc()` returns a character, as an unsigned converted to an `int`, from the file and advances the associated file position indicator for the stream, if defined. If `getc()` is implemented as a macro, it may evaluate `stream` more than once. Therefore, the parameter should never be an expression with side effects. `getchar()` is a macro that is equivalent to `getc(stdin)`.

`stream` is a pointer to the C I/O `FILE` structure.

`getc()` returns the next character from the input stream pointed to by `stream`. If the stream is at end-of-file, the end-of-file indicator for the stream is set and `getc()` returns `EOF`. If a read error occurs, the error indicator for the stream is set and `getc()` returns `EOF`.

`getchar()` returns the next character from the input stream pointed to by `stdin`. If the stream is at end-of-file, the end-of-file indicator for the stream is set and `getchar()` returns `EOF`. If a read error occurs, the error indicator for the stream is set and `getchar()` returns `EOF`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

fgetc()
fopen()
putc(), putchar()

get_const()

Get Current Constant Data

Syntax

```
#include <regs.h>
void *get_const(void);
```

Description

`get_const()` returns the current constant data pointer.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

get_current_tss()

Get Current Task Segment

Syntax

```
#include <types.h>
#include <regs.h>
u_int32 *get_current_tss(void);
```

Description

`get_current_tss()` returns the current task segment.

Note

`get_current_tss()` is only supported on the 80x86 family of processors version of the compiler.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

getenv()

Get Value for Environment Name

Syntax

```
#include <stdlib.h>
char *getenv(const char *name);
```

Description

`getenv()` searches the environment list for a string that matches the specified name.

`name` is a pointer to the name.

`getenv()` returns a pointer to a string associated with the matched list member. The string pointed to cannot be modified by the program. If the specified name cannot be found, a null pointer is returned.

For More Information

The common set of environment names depends on the operating system. Refer to your operating system's user manual for more information.

The environment list is implemented as a NULL terminated array of pointers to characters. Each string is in the form:

```
<name>=<value>
```

`<name>` is the name of the environment variable.

`<value>` is the current value of the environment variable.

To modify the list, you must either find the existing entry or enlarge the array and add an entry. The base of the array is stored in the external variable `_environ`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`_os_chain()`
`_os_chainm()`
`_os_dfork()`
`_os9_dfork()`
`_os_dforkm()`
`_os_exec()`
`_os_fork()`
`os9fork(), os9forkc()`
`_os_forkm()`

getgid()

Return Group ID

Syntax

```
#include <UNIX/os9def.h>
unsigned int getgid();
```

Description

`getgid()` returns the group identification of the current process.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

get_excpt_base()

Return Exception Table Base Address

Syntax

```
#include <types.h>
#include <regs.h>
u_int32 *get_excpt_base(void);
```

Description

`get_excpt_base()` returns the exception table base address.

Note

`get_excpt_base()` is only supported on the 80x86 family of processors version of the compiler.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

get_gdtr()

Get Global Descriptor Pointer

Syntax

```
#include <types.h>
#include <regs.h>
void *get_gdtr(void);
```

Description

`get_gdtr()` returns a pointer to the global descriptor.

Note

`get_gdtr()` is only supported on the 80x86 family of processors version of the compiler.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

getime()

Get System Time

Syntax

```
#include <time.h>
int getime(struct sgtbuf *timebuf);
```

Description

`getime()` returns the system time in a time buffer. The time units are defined in the `time.h` header file.

`timebuf` is a pointer to the time buffer.

If successful, `getime()` returns the buffer pointer.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_getime()`	
`F$Time`	***OS-9 for 68K Technical Manual***
`F_TIME`	***OS-9 Technical Manual***

_get_module_dir()

Get Module Directory Entry

Syntax

```
#include <module.h>
int _get_module_dir(
     void         *buffer,
     int          count);
```

Description

`_get_module_dir()` copies the system's module directory into the buffer for inspection. A maximum of `count` bytes are copied. `_get_module_dir()` returns the number of bytes actually copied.

`buffer` is a pointer to the buffer.

`count` specifies the maximum number of bytes to copy.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_get_moddir()`

`F$GModDr` ***OS-9 for 68K Technical Manual***

getnextpwnam()

Get Next Password File Entry for Name

Syntax

```
#include <UNIX/pwd.h>
struct passwd *getnextpwnam(char *name);
```

Description

`getnextpwnam()` returns the next password file entry associated with `name`. `getnextpwnam()` returns NULL if no more password file entries are found for the `name`.

This function is useful for OS-9 systems where users have more than one password file entry.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

See Also

`endpwent()`
`fgetpwent()`
`getpwent()`
`getpwuid()`
`getpwnam()`
`setpwent()`

getopt()

Return Next Option Letter in String

Syntax

```
#include <UNIX/os9def.h>
extern char *optarg;
extern int optind, opterr;
int getopt(
     int          argc,
     char         **argv,
     char         *optstring);
```

Description

`getopt()` returns the next option letter in `argv` that matches a letter in `optstring`. `optstring` must contain the option letters the command using `getopt()` recognizes. If a letter is followed by a colon, the option is expected to have an argument, or a group of arguments, which must be separated from it by white space.

`optarg` is set to point to the start of the option argument on return from `getopt()`.

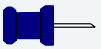
`getopt()` places in `optind` the `argv` index of the next argument to be processed. `optind` is external and is initialized to `1` before the first call to `getopt()`.

When all options have been processed (up to the first non-option argument), `getopt()` returns `-1`. The special option `--` may be used to delimit the end of the options. When it is encountered, `-1` is returned and `--` is skipped.

`getopt()` prints an error message on the standard error and returns a question mark (?) when it encounters an option letter not included in `optstring` or no option-argument after an option that expects one. This error message may be disabled by setting `opterr` to `0`.

For example, this code fragment shows how one might process the arguments for a command that can take the mutually exclusive options a and b, and the option o, which requires an option argument:

```
main(argc, argv);
int argc;
char **argv;
{
int c;
extern char *optarg;
extern int optind;
.
.
.
while ((c = getopt(argc, argv, “abo:”)) != -1)
   switch (c) {
     case ‘a’:
         if (bflg)
            errflg++;
         else
            aflag++;
         break;
     case ‘b’:
         if (aflg)
            errflg++;
         else
            bproc ();
         break;
     case ‘o’:
         ofile = optarg;
         break;
     case ‘?’:
         errflg++;
   }
if (errflg) {
   (void)fprintf(stderr, “usage: . . .”);
   exit (2);
}
for (; optind < argc; optind++) {
    if (access(argv[optind], 4)) {
       .
       .
       .
     }
  }
```



Note

Changing the value of the variable `optind`, or calling `getopt()` with different values of `argv`, may lead to unexpected results.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Unsafe
Re-entrant:	No

Library

`unix.l`

getpid()

Return Process ID Number

Syntax

```
#include <process.h>
int getpid(void);
```

Description

`getpid()` returns the system process ID number for the calling process. You can use this number for tasks such as creating unique file names.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_id()`	
`_os9_id()`	
`F$ID`	***OS-9 for 68K Technical Manual***
`F_ID`	***OS-9 Technical Manual***

getppid()

Return Parent Process ID Number

Syntax

```
#include <UNIX/os9def.h>
unsigned int getppid(void);
```

Description

`getppid()` returns the system process ID number for the parent of the calling process. You can use this number for tasks such as creating unique file names.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

See Also

`getpid()`

_get_process_desc()

Get Process Descriptor Copy

Syntax

```
#include <process.h>
int _get_process_desc(
     int          pid,
     int          count,
     pr_desc      *buffer);
```

Description

`_get_process_desc()` copies a process descriptor into the buffer for inspection.

`pid` is the process ID number of the desired process. Only the lower 16 bits of `pid` are used.

`count` specifies the maximum number of bytes to copy.

`buffer` is a pointer to the buffer.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

The following is an example call:

```
procid pbuf;
_get_process_desc(pid,sizeof(pbuf),&pbuf);
```

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

_os_gprdsc()
F$GPrDsc *OS-9 for 68K Technical Manual*

_get_process_table()

Get Process Table Entry

Syntax

```
#include <process.h>
int _get_process_table(
     void         *buffer,
     int          count);
```

Description

`_get_process_table()` copies the system's process descriptor table into the buffer for inspection. A maximum of `count` bytes are copied. The number of bytes actually copied is returned.

`buffer` is a pointer to the buffer.

`count` specifies the maximum number of bytes to copy.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_get_prtbl()`
`F$GPrDBT` ***OS-9 for 68K Technical Manual***

getpw()

Get Name from Password File

Syntax

```
#include <UNIX/pwd.h>
int getpw(
     int          uid,
     char         *buf);
```

Description

`getpw()` searches the password file for the (numerical) user identification number (`uid`) and fills in the buffer (`buf`) with the corresponding name. It returns non-zero if the `uid` could not be found. The name is null-terminated.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

See Also

`endpwent()`
`fgetpwent()`
`getpwent()`
`getpwuid()`
`getpwnam()`
`setpwent()`

getpwent()

Get Password File Entry

Syntax

```
#include <UNIX/pwd.h>
struct passwd *getpwent(void);
```

Description

`getpwent()`, `getpwuid()` and `getpwnam()` each return a pointer to an object with the following structure containing the fields of a line in the password file. Each line in the file contains a `passwd` structure, declared in the `<UNIX/pwd.h>` header file:

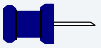
```
struct    passwd {
   char *pw_name;    * login name
   char *pw_passwd;  * password
   int pw_uid;       * user ID
   int pw_gid;       * group ID
   int  pw_prio;     * priority
   char *pw_age;     * unsupported -zero length string
   char *pw_comment; * unsupported -zero length string
   char *pw_gecos;   * unsupported -zero length string
   char *pw_dir;     * initial data directory
   char *pw_xdir;    * initial execution directory
   char *pw_shell;   * initial command executed
   char *pw_junk;    * undefined
};
```

The fields `pw_age`, `pw_comment`, `pw_gecos`, and `pw_junk` are unused. When first called, `getpwent()` returns a pointer to the first `passwd` structure in the file; thereafter, it returns a pointer to the next `passwd` structure in the file; so successive calls can be used to search the entire file. `getpwuid()` searches from the beginning of the file until a numerical user ID matching `uid` is found and returns a pointer to the particular structure in which it was found. `getpwnam()` searches from the beginning of the file until a `login` name matching `name` is found, and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

A call to `setpwent()` has the effect of rewinding the password file to allow repeated searches. `endpwent()` may be called to close the password file when processing is complete.

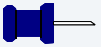
`fgetpwent()` returns a pointer to the next `passwd` structure in the stream `f`, which matches the format of the password file `/dd/SYS/password`.

`getpwent()`, `getpwuid()`, and `getpwnam()` return a pointer to `struct passwd` on success. On EOF or error, or if the requested entry is not found, they return NULL.



Note

The above routines use the standard I/O library, which increases the size of programs not otherwise using standard I/O more than might be expected.



Note

All information is contained in a static area which is overwritten by subsequent calls to these functions, so it must be copied if it is to be saved.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

See Also

endpwent()
getpwent()
getpw()
getpwuid()
getpwnam()
setpwent()

getpwnam()

Get Password File Name

Syntax

```
#include <UNIX/pwd.h>
struct passwd *getpwnam(char *nam);
```

Description

`getpwnam()` returns a pointer to an object with the structure shown in `getpwent()` containing the fields of a line in the password file that matches the name `nam`. Each line in the file contains a `passwd` structure, declared in the `<UNIX/pwd.h>` header file.

Note

The above routines use the standard I/O library, which increases the size of programs not otherwise using standard I/O more than might be expected.

Note

All information is contained in a static area which is overwritten by subsequent calls to these functions, so it must be copied if it is to be saved.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

unix.l

See Also

endpwent()
fgetpwent()
getpw()
getpwent()
getpwuid()
setpwent()

getpwuid()

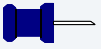
Get Password File User ID

Syntax

```
#include <UNIX/pwd.h>
struct passwd *getpwuid(unsigned int uid);
```

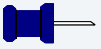
Description

`getpwuid()` returns a pointer to an object with the structure shown in `getpwent()`containing the fields of a line in the password file that matches the specified user identification (uid). Each line in the file contains a `passwd` structure, declared in the `<UNIX/pwd.h>` header file.



Note

The above routines use the standard I/O library, which increases the size of programs not otherwise using standard I/O more than might be expected.



Note

All information is contained in a static area which is overwritten by subsequent calls to these functions, so it must be copied if it is to be saved.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

unix.l

See Also

endpwent()
fgetpwent()
getpw()
getpwent()
getpwnam()
setpwent()

gets()

Get String from File

Syntax

```
#include <stdio.h>
char *gets(char *str);
```

Description

`gets()` reads characters from the standard input file into an array until end-of-file is encountered or a newline character is read. Any newline character is discarded, and a null character is written immediately after the last character read into the array.

`str` is a pointer to the array.

`gets()` returns `str` if successful. If end-of-file is encountered and no characters have been read into the array, the contents of the array remain unchanged and a null pointer is returned. If a read error occurs during the operation, the array contents are indeterminate and a null pointer is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fgets()`

getstat()

Get File Status

Syntax

```
#include <sgstat.h>
#include <sg_codes.h>

int getstat(
     int          code,
     int          path,
     struct sgbuf *buffer);      * code = 0

     int          getstat(
     int          code,
     int          path);         * code = 1 or 6

     int          getstat(
     int          code,
     int          path,
     long         *size);        * code = 2

     int          getstat(
     int          code,
     int          path,
     long         *pos);         * code = 5
```

Description

`getstat()` is historical from the 6809 C Compiler. It accesses a few `I$GetStt` system functions.

code is defined as follows:

Table 2-13 getstat() Code Values

Code	Description
0	buffer is the address of the 128 byte buffer into which the path option bytes are copied. The sgstat.h header file contains a struct defined for use by the program.
1	This applies only to SCF devices and is used to test for data ready. The return value is the number of bytes available or -1 if an error occurred.
2	size is the address of the long integer into which the current file size is placed. The function returns -1 on error and zero on success.
5	pos is the address of the long integer into which the current file position is placed. The function returns -1 on error and zero on success.
6	The function returns -1 on either EOF or error, and 0 if not at EOF.

Note

Only the lower 16 bits of code are used.

path must be the path number of an open file.

When getstat() returns with the value -1, the appropriate error code is placed in the global variable errno.

Note

`getstat()` is supported for the convenience of programs ported from the 6809.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_gs_devnm()`
`_os_gs_eof()`
`_os_gs_fd()`
`_os_gs_fdinf()`
`_os_gs_popt()`
`_os_gs_pos()`
`_os_gs_ready()`
`_os_gs_size()`
`I$GetStt` ***OS-9 for 68K Technical Manual***

get_static()

Return Current Static Storage Pointer

Syntax

```
#include <regs.h>
void *get_static(void);
```

Description

`get_static()` returns a pointer to the current static storage.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

See Also

`change_static()`

_getsys()

Get System Global Variables

Syntax

```
#include <sysglob.h>
#include <setsys.h>
int _getsys(
     int          glob,
     int          size);
```

Description

`_getsys()` examines a system global variable. These variables are defined in `setsys.h` for C programs. Each of these variables begin with a `D_` prefix.

`glob` is the system global variable to examine. Only the lower 16 bits of `glob` are used.

`size` is the size of the variable.

`_getsys()` returns the value of the variable if the examine request succeeds. If the request fails, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_getsys()`
`_os_setsys()`
`F$SetSys` ***OS-9 for 68K Technical Manual***

gettimeofday()

Get Current Time

Syntax

```
#include <UNIX/os9time.h>
int gettimeofday(
     struct timeval *tp,
     struct timezone *tzp);
```

Description

The system's notion of the current time is obtained with the `gettimeofday()` call. The current time is expressed in elapsed seconds and microseconds since January 1, 1970 (zero hour). The resolution of the system clock is hardware dependent. The time may be updated continuously, or in `"ticks."`

`gettimeofday()` returns zero (`0`) on success and `-1` on failure and sets `errno` to indicate the error.

`tp` points to a `timeval` structure, which includes the following members:

```
long tv_sec;      * seconds since Jan. 1, 1970
long tv_usec;     * and microseconds
```

If `tp` is a NULL pointer, the current time information is not returned. `tzp` is not supported by `unix.l`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

getuid()

Determine User ID Number

Syntax

```
#include <process.h>
int getuid(void);
```

Description

`getuid()` returns the group/user ID of the current process. The upper word (two bytes) of the value is the group number; the lower word is the user number.

If unsuccessful, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_id()`	
`_os9_id()`	
`F$ID`	***OS-9 for 68K Technical Manual***
`F_ID`	***OS-9 Technical Manual***

getw()

Read Word from File

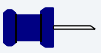
Syntax

```
#include <stdio.h>
int getw(FILE *stream);
```

Description

`getw()` returns a word from a file converted to an integer. It returns `-1` on error. Therefore, use the macros `feof()` and `ferror()` to check the success of `getw()`.

`stream` is a pointer to the C I/O `FILE` structure.



Note

`getw()` is a machine-dependent function because the size of a word varies from machine to machine. `getw()` assumes no particular alignment in the file. `getw()` reads two bytes from the file and sign-extends them to four bytes.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

`feof()`
`ferror()`

getwd()

Get Working Directory Path Name

Syntax

```
#include <UNIX/os9def.h>
char *getwd(char pathname[MAXPATHLEN]);
```

Description

`getwd()` copies the absolute pathname of the current working directory to `pathname` and returns a pointer to the result.

`getwd()` returns NULL and global variable `errno` if an error occurs.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

gmtime()

Convert Calendar Time to Greenwich Mean Time

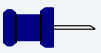
Syntax

```
#include <time.h>
struct tm *gmtime(time_t *tp);
```

Description

`gmtime()` converts the calendar time contained in a time structure to Coordinated Universal Time (UTC). `gmtime()` returns a pointer to that object or a null pointer if UTC is not available.

`tp` is a pointer to the time structure.



Note

`gmtime()` returns a pointer to a static area which may be overwritten. To ensure data integrity, use the structure or save it immediately.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`mktime()` (for more information on the time structure tp)
`time()`

_gregorian()

Convert Date/Time to Gregorian Value

Syntax

```
#include <time.h>
int _gregorian(int *time, int *date);
```

Description

`_gregorian()` converts the time and date from Julian format to the Gregorian equivalents.

`time` and `date` are pointers to the Julian time and date values. The values must be in the following format:

Figure 2-2 _gregorian() Input Date and Time Format

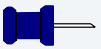
Time:	Seconds Since Midnight (0-86399)
Date:	Julian Day Number

`_gregorian()` modifies the values to the Gregorian format:

Figure 2-3 _gregorian() Output Date and Time Format

Time:	0	Hour (0-23)	Minute	Second
Date:	Year (2-Bytes)		Month	Day
	Byte 0	**Byte 1**	**Byte 2**	**Byte 3**

If successful, `_gregorian()` returns zero. Otherwise, `-1` is returned.



Note

Be careful to pass pointers for the `date` and `time` values.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_julian()` (OS-9 for 68K)
`_os_gregorian()` (OS-9 for 68K)

_gs_devn()

Get Device Name

Syntax

```
#include <sg_codes.h>
int _gs_devn(int path, char *buffer);
```

Description

`_gs_devn()` allows you to determine the name of the device open on a path.

`path` is the path number of an open path.

`buffer` is a pointer to a character array into which the null-terminated device name is placed.

If the path number is invalid, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Note

You must reserve a buffer of at least 32 bytes to receive the device name. Some networked devices using NFM should have a buffer size of 128 or 256 bytes for the `_gs_devn()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_gs_devnm()`	
`I$GetStt`	***OS-9 for 68K Technical Manual***
`I_GETSTAT`, `SS_DEVNAME`	***OS-9 Technical Manual***

_gs_eof()

Check for End of File

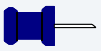
Syntax

```
#include <sg_codes.h>
int _gs_eof(int path);
```

Description

`_gs_eof()` determines if the file open on `path` is at the end of file. If it is at the end of file, `1` is returned. Otherwise, `0` is returned.

`path` specifies the path number of the open file. If `path` is invalid, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

`_gs_eof()` cannot determine end-of-file on SCF devices. SCF devices return an `EOS_EOF` error when the EOF character is read. Do not use this call if you are using the buffered I/O facility on the path. Instead, use `feof()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

feof()	
_os_gs_eof()	
I$GetStt	***OS-9 for 68K Technical Manual***
I_GETSTAT	***OS-9 Technical Manual***
SS_EOF	***OS-9 Technical Manual***

_gs_gfd()

Get File Descriptor Sector

Syntax

```
#include <rbf.h>
int _gs_gfd(
     int          path,
     struct fd    *buffer,
     int          count);
```

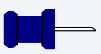
Description

`_gs_gfd()` places a copy of the RBF file descriptor sector of the file open on `path` into a buffer. The structure `fd`, declared in the `rbf.h` header file, provides a convenient means to access the file descriptor information.

`path` specifies the path number of the open file.

`buffer` is a pointer to the buffer in which to place the file descriptor information.

`count` specifies the maximum number of bytes to copy. If an error occurs, `-1` is returned and the appropriate error value is placed in the global variable `errno`.



Note

Be sure the buffer is large enough to hold `count` bytes. A file descriptor is one sector in length and a sector may be between 256 bytes and 32K in size. If an entire descriptor is desired, you must first find the sector size on the device and allocate a buffer. The sector size can be found in the path descriptor's option section. See `_os_gs_popt()` and your operating system technical manual for more information. Generally, the `struct fd` in `rbf.h` is only large enough to hold a 256-byte file descriptor.

Note

This call is effective only on RBF devices.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_gs_fd()`
`_os_gs_popt()`
`_os_ss_fd()`
`I$GetStt` — ***OS-9 for 68K Technical Manual***
`I_GETSTAT`, `SS_FD` — ***OS-9000 Technical Manual***

_gs_opt()

Get Path Options

Syntax

```
#include <sg_codes.h>
int _gs_opt(
     int          path,
     struct sgbuf *buffer);
```

Description

`_gs_opt()` copies the options section of the path descriptor open on `path` into a buffer. `sgbuf` provides a convenient means to access the individual option values. `sgbuf` is declared in the `sgstat.h` header file.

`path is` the path number of the open path descriptor.

`buffer` is a pointer to the buffer.

If the path is invalid, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Note

Be sure the buffer is large enough to hold the options. Declaring the buffer as type `struct sgbuf` is safe because this structure is predefined to be large enough for the options.

Note

If you use `_gs_opt()` on OS-9, you must use the OS-9 for 68K header file. `_gs_opt()` is provided on OS-9 solely for compatibility with OS-9 for 68K. If the OS-9 for 68K header file is unavailable, change the source code to use `_os_gs_popt()` or `_os_gs_luopt()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_gs_popt()`	
`_os_ss_popt()`	
`I$GetStt`	***OS-9 for 68K Technical Manual***
`I_GETSTAT`	***OS-9 Technical Manual***
`SS_PATHOPT`	***OS-9 Technical Manual***
`tmode`	***Using OS-9 for 68K*** or ***Using OS-9***

_gs_pos()

Get Current File Position

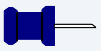
Syntax

```
#include <sg_codes.h>
int _gs_pos(int path);
```

Description

`_gs_pos()` returns the value of the file pointer for the file open on `path`.

`path` is the path number of the open file. If the path is invalid or the device is not an RBF device, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

This call is effective only on RBF devices. Do not use this call if buffered I/O is being performed on the path; instead use `ftell()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`ftell()`	
`_os_gs_pos()`	
`I$GetStt`	***OS-9 for 68K Technical Manual***
`I_GETSTAT, SS_POS`	***OS-9 Technical Manual***

_gs_rdy()

Test for Data Available

Syntax

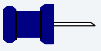
```
#include <sg_codes.h>
int _gs_rdy(int path);
```

Description

`_gs_rdy()` checks the device open on `path` to see if data is waiting to be read. Read requests to the operating system wait until enough bytes have been read to satisfy the specified byte count, thereby suspending the program until the read is satisfied.

`_gs_rdy()` can determine the number of bytes, if any, waiting to be read, and then issue a read request for only the number of bytes actually available.

If the path is invalid or no data is available to be read, `-1` is returned, and the appropriate error code is placed in the global variable `errno`. Otherwise, the number of bytes available to be read is returned.



Note

This call is effective only on RBF, SCF, or pipe devices. `_gs_rdy()` is not intended for use with the buffered I/O calls such as `getc()`; unpredictable results may occur. Use low-level functions when using `_gs_rdy()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

_os_gs_ready()
_os_read()
_os_readln()

I$GetStt	***OS-9 for 68K Technical Manual***
I_GETSTAT	***OS-9 Technical Manual***
SS_READY	***OS-9 Technical Manual***

_gs_size()

Get Current File Size

Syntax

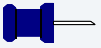
```
#include <sg_codes.h>
int _gs_size(int path);
```

Description

`_gs_size()` determines the current size of the file open on `path`.

`path` is the path number of the open file.

If the path is invalid or the device is not an RBF device, `-1` is returned and the appropriate error code is placed in the global variable `errno`. Otherwise, the file size is returned.



Note

This call is effective only on RBF and PIPEMAN devices.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_gs_size()`	
`I$GetStt`	***OS-9 for 68K Technical Manual***
`I_GETSTAT`	***OS-9 Technical Manual***
`SS_SIZE`	***OS-9 Technical Manual***

Syntax

```
#include <math.h>
double hypot(double x, double y);
```

Description

`hypot()` returns the Euclidean distance function:

```
sqrt(x * x + y * y)
```

`hypot()` ignores overflows.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

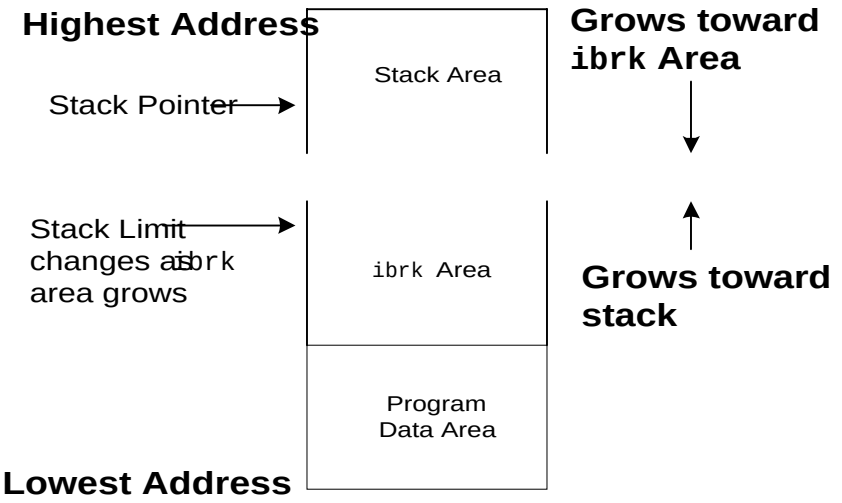
Syntax

```
#include <stdlib.h>
char *ibrk(unsigned size);
```

Description

`ibrk()` returns a pointer to a memory block. The returned pointer is aligned to a word boundary. The memory from which `ibrk()` grants requests is the area between the end of the data allocation and the stack:

Figure 2-4 ibrk() Memory Stack Diagram

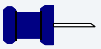


`size` specifies the size, in bytes, of the memory block.

If the requested size would cause the `ibrk` area to cross the stack pointer, the request fails. You can use `freemem()` to determine the amount of stack remaining which is also the remaining `ibrk` area.

`ibrk()` is useful to request memory from a fixed-size area of memory, unlike `ebrk()` whose available memory is that of the entire system. The C I/O library functions request the first 2K of I/O buffers from this area, the remainder from `ebrk()`.

If successful, `ibrk()` returns zero. Otherwise, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

Be very careful not to crowd out the stack with `ibrk()` calls. When stack checking is in effect, the program aborts with a `***Stack Overflow***` message if insufficient stack area exists to call a function.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`ebrk()`
`_freemem(), freemem()`
`sbrk()`
`stacksiz(), _stacksiz()`

inc()

Get Character From I/O Address

Syntax

```
#include <regs.h>
#include <types.h>
u_char inc(void *address);
```

Description

`inc()` returns the character indicated by the specified I/O address on the 80x86 processors or the memory address on all other processors.

`address` is the specified I/O address.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

See Also

`inl()`
`inw()`
`outc()`
`outl()`
`outw()`

Syntax

```
#include <strings.h>
char *index(
     char         *ptr,
     int          ch);
```

Description

`index()` returns a pointer to the first occurrence of a character in a string. If the character is not found, `index()` returns NULL.

`ptr` is a pointer to the string.

`ch` is the character for which to search. Only the lower 8 bits of `ch` are used.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

Example

This example looks for a period (`.`) in `string` and sets the pointer `ptr`. Note that `ch` is a character, not a pointer to a character:

```
func()
{
   char *ptr,*string;
   if((ptr = index(string,'.'))
      process(ptr);
   else printf("No '.' found!\n");
}
```

See Also

rindex()
strchr()
strrchr()

inl()

Get Integer from I/O Address

Syntax

```
#include <regs.h>
#include <types.h>
u_int32 inl(void *address);
```

Description

`inl()` returns the integer indicated by the specified I/O address on the 80x86 processors or the memory address on all other processors.

`address` is the specified I/O address.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

See Also

`inc()`
`inw()`
`outc()`
`outl()`
`outw()`

intercept()

Set Up Process Signal Handler

Syntax

```
#include <signal.h>
int intercept(void (*icpthand)(int));
```

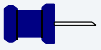
Description

`intercept()` instructs the operating system to pass control to a signal handler function when the process receives a signal.

`icpthand` is a pointer to the function.

If the signal handler function declares an `int` parameter, the function has access to the value of the received signal. On return from the signal handler function, the process resumes at the point in the program where it was interrupted by the signal.

If `icpthand` is zero, the signal handler is removed.



Note

A program does not receive an abort or quit signal from the keyboard (usually ^C and ^E) unless the program performs output to the terminal. This is because the operating system sends the abort/quit signals to the last process to perform I/O to the terminal. If you run the program from the shell and type ^E before the program performs I/O to the terminal, the shell receives the signal and kills the running program. If a program requires immediate control of the terminal, perform some I/O to one of the standard paths such as printing a program banner or getting the terminal options with `_gs_opt()`.

Any I/O using the C library (for example, `printf()`) cannot be performed inside both the intercept handler function and the main program. Consequently, avoid I/O within intercept functions.

Note

Generally, `intercept()` should not be used with `signal()`. Only signals that are under `SIG_DFL` handling reach a handler installed with this function.

You cannot use floating point math within the intercept handler function.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_intercept()`	
`_os_send()`	
`_os_sigmask()`	
`signal()`	
`F$Icpt`	***OS-9 for 68K Technical Manual***
`F$Send`	***OS-9 for 68K Technical Manual***
`F$SigMask`	***OS-9 for 68K Technical Manual***
`F_ICPT`	***OS-9 Technical Manual***
`F_SEND`	***OS-9 Technical Manual***
`F_SIGMASK`	***OS-9 Technical Manual***

Example

As an example, this program is designed to clean up work files and exit when it receives a keyboard quit signal (signal 2 which, normally, is caused by typing ^E on the terminal):

```
#include <stdio.h>

char *tempname = "tempfile";
FILE *tempfp;
int quittime = 0;

/* the signal handler */
gotsignl(signum)
int signum;
{
   switch(signum) {
      case 2:                * the quit signal
         quittime = 1;
         break;
      default:               * ignore all others
         break;
   }
}
main()
{
   if((tempfp = fopen(tempname,"w") == NULL)
      exit(_errmsg(errno,
            "can't open file - %s\n",tempname));
   intercept(gotsignl);
   do {
      do_work();
   } while(!quittime);
   fclose(tempfp);
   unlink(tempname);
   exit(_errmsg(1,"quittin' time!!!\n"));
}
```

Syntax

```
#include <ioctl.h>
#include <UNIX/os9def.h>
int ioctl(
     unsigned    int fd,
     unsigned    int request,
     caddr_t     arg);
```

Description

`ioctl()` performs some of the UNIX-like I/O control functions.

`fd` is the path to the device that is affected by the function.

`request` is the function to perform.

`arg` has various meanings depending on the function to be performed.

The two functions that are supported are:

- `FIONREAD` — Return the number of bytes available for reading from the path. The number of bytes is returned at the `unsigned long` pointed to by `arg`. `FIONREAD` works with `RBF, SCF, Pipe`, and `Sockman`.
- `FIONBIO` — Set the I/O blocking status on the path. `arg` is a pointer to `0` (blocking I/O) or `1` (non-blocking I/O). `FIONBIO` works with `Sockman` (only `path_opts` block type it gets and checks for).

`ioctl()` returns zero on success for most requests. Some specialized requests may return non-zero values on success. See the specific description for the request. On failure, `ioctl()` returns `-1` and sets `errno` to indicate an error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

unix.l

inw()

Get Word from I/O Address

Syntax

```
#include <regs.h>
#include <types.h>
u_int16 inw(void *address);
```

Description

`inw()` returns the word indicated by the specified I/O address on the 80x86 processors or the memory address on all other processors.

`address` is the specified address.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

See Also

`inc()`
`inl()`
`outc()`
`outl()`
`outw()`

irq_change()

Set IRQ Level

Syntax

```
#include <regs.h>
#include <types.h>
status_code irq_change(status_code newval);
```

Description

`irq_change()` sets the interrupt level to `newval`. `irq_change()` returns the previous interrupt level. It is a system-state-only system call.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

See Also

`irq_disable()`
`irq_enable()`
`irq_maskget()`
`irq_restore()`
`irq_save()`

Syntax

```
#include <regs.h>
void irq_disable(void);
```

Description

`irq_disable()` masks interrupts to the highest interrupt level available on the CPU. It is a system-state-only system call.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

See Also

`irq_change()`
`irq_enable()`
`irq_maskget()`
`irq_restore()`
`irq_save()`

irq_enable()

Unmask Interrupts

Syntax

```
#include <regs.h>
void irq_enable(void);
```

Description

`irq_enable()` unmasks all interrupts to the lowest interrupt level available on the CPU.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

See Also

`irq_change()`
`irq_disable()`
`irq_maskget()`
`irq_restore()`
`irq_save()`

irq_maskget()

Mask Interrupts; Return Previous Level

Syntax

```
#include <regs.h>
status_code irq_maskget (void);
```

Description

`irq_maskget()` masks all interrupts to the highest level available on the CPU. `irq_maskget()` returns the interrupt level before masking.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

See Also

`irq_change()`
`irq_disable()`
`irq_enable()`
`irq_restore()`
`irq_save()`

irq_restore()

Restore Interrupt Level

Syntax

```
#include <regs.h>
void irq_restore(status_code stat_reg);
```

Description

`irq_restore()` sets the interrupt level.

`stat_code` specifies the value to put in the processor status register.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

See Also

`irq_change()`
`irq_disable()`
`irq_enable()`
`irq_maskget()`
`irq_save()`

irq_save()

Return Current Interrupt Level

Syntax

```
#include <regs.h>
status_code irq_save(void);
```

Description

`irq_save()` returns the current value of the status register.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

See Also

`irq_change()`
`irq_disable()`
`irq_enable()`
`irq_maskget()`
`irq_restore()`

isalnum()

See If Parameter Is Alphanumeric

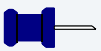
Syntax

```
#include <ctype.h>
int isalnum(int c);
```

Description

`isalnum()` returns a non-zero value for any character for which `isalpha()` or `isdigit()` is true.

c is the character to test.



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isalnum)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

See Also

isalpha()
iscntrl(),
isdigit()
isgraph()
islower()
isprint()
ispunct()
isspace()
isupper()
isxdigit()

isalpha()

See If Parameter Is Alphabetic

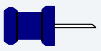
Syntax

```
#include <ctype.h>
int isalpha(int c);
```

Description

`isalpha()` returns a non-zero value for any character for which `isupper()` or `islower()` is non-zero. `isalpha()` returns non-zero only for the characters for which `isupper()` or `islower()` is non-zero.

c is the character to test.



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isalpha)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`isalnum()`
`iscntrl()`
`isdigit()`
`isgraph()`
`islower()`
`isprint()`
`ispunct()`
`isspace()`
`isupper()`
`isxdigit()`

_isascii(), isascii()

See If Parameter Is ASCII

Syntax

```
#include <ctype.h>
int _isascii(int c);
int isascii(int c);
```

Description

`_isascii()` returns a non-zero value if c is an ASCII character (the value is in the range `0x00` through `0x7f`). Otherwise, it returns zero. Only `_isascii()` is available in strictly conforming ANSI mode. Otherwise, both are available. `_isascii()` and `isascii()` are macro definitions.

c is the character to test.

Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isascii)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`ctype.h`

See Also

`isalnum()`
`isalpha()`
`iscntrl()`
`isdigit()`
`isgraph()`
`islower()`
`isprint()`
`ispunct()`
`isspace()`
`isupper()`
`isxdigit()`

isatty()

Find Terminal Name

Syntax

```
#include <UNIX/os9def.h>
#include <setsys.h>
int isatty(int fd);
```

Description

`isatty()` returns `1` if `fd` is associated with a SCF device. Otherwise, it returns zero.

Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`. The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isatty)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

iscntrl()

See If Parameter Is Control Code

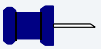
Syntax

```
#include <ctype.h>
int iscntrl(int c);
```

Description

`iscntrl()` is a C macro that returns a non-zero value for ASCII characters with values from `0` to `0x1f` and `0x7f`.

c is the character to test.



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (iscntrl)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

ctype.h

See Also

isalnum()
isalpha()
isdigit()
isgraph()
islower()
isprint()
ispunct()
isspace()
isupper()
isxdigit()

isdigit()

See If Parameter Is Digit

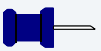
Syntax

```
#include <ctype.h>
int isdigit(int c);
```

Description

`isdigit()` is a C macro that returns a non-zero value for any decimal-digit character.

c is the character to test.



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isdigit)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

ctype.h

See Also

isalnum()
isalpha()
iscntrl()
isgraph()
islower()
isprint()
ispunct()
isspace()
isupper()
isxdigit()

_iseuc_kana()

See If Parameter Is the First Byte of a Half Width Kaka-Kana Character

Syntax

```
#include <ctype.h>
int _iseuc_kana(int c);
```

Description

`_iseuc_kana()` is a C macro that returns a non-zero value if `c` is the first byte of a half-width Kata-Kana character. Otherwise, it returns zero. The first byte of a half-width Kata-Kana character is `0X8E`.

`c` is the character to test.

Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (iseuc_kana)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

ctype.h

See Also

isalnum()
isalpha()
iscntrl()
isdigit()
isgraph()
islower()
isprint()
ispunct()
isspace()
isupper()
isxdigit()

Syntax

```
#include <ctype.h>
int _iseuc1();
```

Description

`_iseuc1()` is a C macro that returns a non-zero value if `C` is the first byte of an `EUC` packed format `JIS x 0208-1990` character. Otherwise, it returns zero. The valid range is between `0xa1` and `0xfe`.

`_iseuc2()` is the same thing for the second byte of the character and has the same valid range.

`c` is the character to test.

Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (iseuc1)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`ctype.h`

See Also

`isalnum()`
`isalpha()`
`iscntrl()`
`isdigit()`
`isgraph()`
`islower()`
`isprint()`
`ispunct()`
`isspace()`
`isupper()`
`isxdigit()`

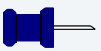
Syntax

```
#include <ctype.h>
int isgraph(int c);
```

Description

`isgraph()` is a C macro that returns a non-zero value if c is any printing character except a space. Otherwise, it returns zero.

c is the character to test.



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isgraph)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

ctype.h

See Also

isalnum()
isalpha()
iscntrl()
isdigit()
islower()
isprint()
ispunct()
isspace()
isupper()
isxdigit()

Syntax

```
#include <ctype.h>
int _isjis_kana(int c);
```

Description

`_isjis_kana()` is a C macro that returns a non-zero value if c is a Hankaku-Kana. Otherwise, it returns zero. The valid range is between `0xa1` and `0xdf`.

c is the character to test.

Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isjis_kana)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

ctype.h

See Also

isalnum()
isalpha()
iscntrl()
isdigit()
isgraph()
islower()
isprint()
ispunct()
isspace()
isupper()
isxdigit()

islower()

See If Parameter Is Lowercase

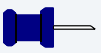
Syntax

```
#include <ctype.h>
int islower(int c);
```

Description

`islower()` is a C macro that tests for lowercase letters. `islower()` returns a non-zero value for ASCII characters from `0x61 (a)` to `0x7a (z)`.

c is the character to test.



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`. The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (islower)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

ctype.h

See Also

isalnum()
isalpha()
iscntrl()
isdigit()
isgraph()
isprint()
ispunct()
isspace()
isupper()
isxdigit()

isprint()

See If Parameter Is Printable Character

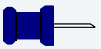
Syntax

```
#include <ctype.h>
int isprint(int c);
```

Description

`isprint()` is a C macro that tests for printing characters. It returns a non-zero value for ASCII characters with values from `0x20` to `0x7e`.

c is the character to test.



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isprint)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

ctype.h

See Also

isalnum()
isalpha()
iscntrl()
isdigit()
isgraph()
islower()
ispunct()
isspace()
isupper()
isxdigit()

ispunct()

See If Parameter Is Punctuation Character

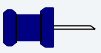
Syntax

```
#include <ctype.h>
int ispunct(int c);
```

Description

`ispunct()` is a C macro that returns a non-zero value for any printing character that is neither a space nor a character for which `isalnum()` is non-zero.

c is the character to test.



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`. The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (ispunct)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

ctype.h

See Also

isalnum()
isalpha()
iscntrl()
isdigit()
isgraph()
islower()
isprint()
isspace()
isupper()
isxdigit()

_issjis1()

See If Parameter Is First Byte of SJIS Kanji

Syntax

```
#include <ctype.h>
int _issjis1(int c);
```

Description

`_issjis1()` is a C macro that returns a non-zero value if `c` is the first byte of a Shift-JIS (Japanese Industrial Standard) Kanji character. Otherwise, it returns zero. The valid ranges for Shift-JIS first bytes are between `0x81` and `0x9f` or between `0xe0` and `0xfc`.

`c` is the character to test.

Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (issjis1)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`ctype.h`

See Also

`isalnum()`
`isalpha()`
`iscntrl()`
`isdigit()`
`isgraph()`
`islower()`
`isprint()`
`ispunct()`
`_issjis2()`
`isspace()`
`isupper()`
`isxdigit()`

_issjis2()

See If Parameter Is Second Byte of SJIS Kanji

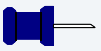
Syntax

```
#include <ctype.h>
int _issjis2(int c);
```

Description

`_issjis2()` is a C macro that returns a non-zero value if `c` is the second byte of a Shift-JIS (Japanese Industrial Standard) Kanji character. Otherwise, it returns zero. The valid range for Shift-JIS second bytes are between `0x40` and `0xfc`.

`c` is the character to test. `c` cannot equal `0x7f`.



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`. The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (issjis2)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

ctype.h

See Also

isalnum()
isalpha()
iscntrl()
isdigit()
isgraph()
islower()
isprint()
ispunct()
_issjis1()
isspace()
isupper()
isxdigit()

Syntax

```
#include <ctype.h>
int isspace(int c);
```

Description

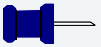
`isspace()` is a C macro that returns a non-zero value for standard white space characters.

c is the character to test.

The standard white space characters are the following:

Table 2-14 isspace() White Space Characters

Description	Character
Space	
Form feed	`\f`
Newline	`\n`
Carriage return	`\r`
Horizontal tab	`\t`
Vertical tab	`\v`



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isspace)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`ctype.h`

See Also

`isalnum()`
`isalpha()`
`iscntrl()`
`isdigit()`
`isgraph()`
`islower()`
`isprint()`
`ispunct()`
`isupper()`
`isxdigit()`

isupper()

See If Parameter Is Uppercase

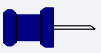
Syntax

```
#include <ctype.h>
int isupper(int c);
```

Description

`isupper()` is a C macro that returns a non-zero value for uppercase letters. `isupper()` returns a non-zero value for ASCII characters with values in the range from `0x41 (A)` to `0x5a (Z)`.

c is the character to test.



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`. The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isupper)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

ctype.h

See Also

isalnum()
isalpha()
iscntrl()
isdigit()
isgraph()
islower()
isprint()
ispunct()
isspace()
isxdigit()

isxdigit()

See If Parameter Is Hexadecimal

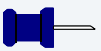
Syntax

```
#include <ctype.h>
int isxdigit(int c);
```

Description

`isxdigit()` is a C macro that returns a non-zero value for any hexadecimal-digit character.

c is the character to test.



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isxdigit)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

ctype.h

See Also

isalnum()
isalpha()
iscntrl()
isdigit()
isgraph()
islower()
isprint()
ispunct()
isspace()
isupper()

Syntax

```
#include <time.h>
int _julian(
     int          *time,
     int          *date);
```

Description

`_julian()` converts the time and date from Gregorian format to the Julian equivalents.

`time` and `date` are pointers to the Gregorian format time and date values. The following format is assumed:

Figure 2-5 _julian() Input Date and Time Formats

Time:	0	Hour (0-23)	Minute	Second
Date:	Year (2-bytes)		Month	Day
	Byte 0	**Byte 1**	**Byte 2**	**Byte 3**

`_julian()` modifies the objects to which its parameters point as follows:

Figure 2-6 _julian() Output Date and Time Formats

Time:	Seconds since midnight (0-86399)
Date:	Julian Day Number

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Note

Be certain to pass pointers for `date` and `time` values.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

Example

```
main()
{
   int date,time,tick;
   short day;
   _sysdate(0,&time,&date,&day,&tick);
   _julian(&time,&date);
   printf("The Julian date is %d. \
      Cinderella needs to be home in %d
seconds!\n",date,86400-time);
}
```

See Also

`_sysdate()` (OS-9 for 68K)
`_os_gregorian()` (OS-9 for 68K)
`_os_julian()` (OS-9 for 68K)
F$Time ***OS-9 for 68K Technical Manual***
F$Julian ***OS-9 for 68K Technical Manual***
F_TIME ***OS-9 Technical Manual***

kill()

Send Signal to Process

Syntax

```
#include <signal.h>
int kill(
     int          pid,
     int          sigcode);
```

Description

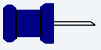
`kill()` sends a signal to a process. Both the sending and receiving process must have the same user number unless the sending process' user number is that of the super user (`0`).

`pid` is the process ID of the process to kill.

`sigcode` is the signal code to send. Only the lower 16 bits of `sigcode` are used.

The value in `sigcode` is sent as a signal to the process specified by `pid`. You can pass any value in `sigcode`. The conventional code numbers are defined in the `signal.h` header file.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

The super user can send a signal to all processes running on the system if the `pid` is zero.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_send()`	
`F$Send`	***OS-9 for 68K Technical Manual*** or
`F_SEND`	***OS-9 Technical Manual***
`kill` command	***Using OS-9 for 68K*** or ***Using OS-9***

labs()

Compute Absolute Value

Syntax

```
#include <stdlib.h>
long int labs(long int num);
```

Description

`labs()` returns the absolute value of `num`. `num` and the returned value each have type `long int`.

`num` is the value to use.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`abs()`

_lcalloc()

Allocate Storage for Array (Low-Overhead)

Syntax

```
#include <stdlib.h>
void *_lcalloc(
     unsigned long nel,
     unsigned long elsize);
```

Description

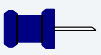
`_lcalloc()` allocates space for an array. The allocated memory is cleared to zeroes.

`nel` specifies the number of elements in the array.

`elsize` specifies the size of each element.

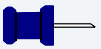
`_lcalloc()` calls `_lmalloc()` to allocate memory.

- If the allocation is successful, `_lcalloc()` returns a pointer to the area.
- If the allocation fails, `_lcalloc()` returns NULL.



Note

Use of the low-overhead allocation functions (`_lcalloc()`, `_lmalloc()`, and `_lrealloc()`) instead of the general allocation functions (`calloc()`, `malloc()`, and `realloc()`) saves 8 bytes per allocation because the low-overhead functions do not save the allocation size or the 4-byte check value.



Note

Use extreme care to ensure that you only access the assigned memory. Modifying addresses immediately above or below the assigned memory causes unpredictable program results.

The low-overhead functions require that you keep track of the sizes of allocated spaces in memory. Note the `_lfree()` and `_lrealloc()` parameters.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`clib.l`

See Also

`calloc()`
`free()`
`_lfree()`
`_lmalloc()`
`_lrealloc()`
`malloc()`
`_mallocmin()`
`_os9_srqmem()`
`realloc()`

ldexp()

Multiply Float by Exponent of 2

Syntax

```
#include <math.h>
double ldexp(
     double       fp,
     int          exp);
```

Description

`ldexp()` multiplies a floating point value by an integral power of two. `ldexp()` returns the value equal to `fp * 2^exp`.

`fp` is the floating point value.

`exp` is the integral power of two.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

ldiv()

Compute Quotient and Remainder

Syntax

```
#include <stdlib.h>
ldiv_t ldiv(
     long int     numer,
     long int     denom);
```

Description

`ldiv()` returns the quotient and remainder of the division of the numerator by the denominator. The parameters and member of the returned structure are all of type `long int`.

`numer` is the numerator.

`denom` is the denominator.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`div()`

_lfree()

Return Memory (Low-Overhead)

Syntax

```
#include <stdlib.h>
void _lfree(
     void         *ptr,
     unsigned long size);
```

Description

`_lfree()` returns a block of memory granted by `_lcalloc()` or `_lmalloc()`. The memory is returned to a pool of memory for later re-use by `calloc()`, `_lcalloc()`, `_lmalloc()`, or `malloc()`.

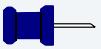
`ptr` is a pointer to the memory block.

`size` specifies the size of the memory block.

`_lfree()` never returns an error.

Note

Use of the low-overhead allocation functions (`_lcalloc()`, `_lmalloc()`, and `_lrealloc()`) instead of the general allocation functions (`calloc()`, `malloc()`, and `realloc()`) saves 8 bytes per allocation because the low-overhead functions do not save the allocation size or the 4-byte check value.



Note

If `_lfree()` is used with something other than a pointer returned by `_lcalloc()` or `_lmalloc()`, the memory lists maintained by `_lmalloc()` and `malloc()` are corrupted and programs behave unpredictably.

The low-overhead functions require that you keep track of the sizes of allocated spaces in memory.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`clib.l`

See Also

`calloc()`
`free()`
`_freemin()`
`_lcalloc()`
`_lmalloc()`
`_lrealloc()`
`malloc()`
`_os_srtmem()`
`realloc()`

lftocr()

Convert Linefeed to Carriage Return

Syntax

```
#include <UNIX/os9def.h>
void lftocr(
     char         *buf,
     int          n);
```

Description

`lftocr()` converts all linefeed characters to carriage-return characters in `buf` for a span of n characters.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

_lmalloc()

Allocate Memory from Arena (Low-Overhead)

Syntax

```
#include <stdlib.h>
void *_lmalloc(unsigned long size);
```

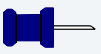
Description

`_lmalloc()` returns a pointer to a memory block. The pointer is suitably aligned for storing data of any type.

`size` specifies the size of the memory block in bytes.

`_lmalloc()` maintains an amount of memory, called an arena, from which it grants memory requests. `_lmalloc()` searches its arena for a block of free memory large enough for the request and, in the process, joins adjacent blocks of free space returned by either `_lfree()` or `free()`. If sufficient memory is not available in the arena, `_lmalloc()` calls `_srqmem()` to get more memory from the system.

`_lmalloc()` returns NULL if no memory is available or if the arena is detected to be corrupted.



Note

Use of the low-overhead allocation functions (`_lcalloc()`, `_lmalloc()`, and `_lrealloc()`) instead of the general allocation functions (`calloc()`, `malloc()`, and `realloc()`) saves 8 bytes per allocation because the low-overhead functions do not save the allocation size or the 4-byte check value.

Note

Use extreme care to ensure that you only access the memory assigned by `_lmalloc()`. Modifying addresses immediately above or below the assigned memory or passing `_lfree()` a value not assigned by `_lmalloc()` causes unpredictable program results.

The low-overhead functions require that you keep track of the sizes of allocated spaces in memory. Note the `_lfree()` and `_lrealloc()` parameters.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`clib.l`

See Also

`_lcalloc()`
`_lfree()`
`_lrealloc()`
`malloc()`
`_mallocmin()`
`calloc()`
`free()`
`_os_srqmem()`
`realloc()`

Syntax

```
#include <locale.h>
struct lconv *localeconv(void);
```

Description

`localeconv()` sets the components of an object with type `struct lconv` with values appropriate for formatting numeric quantities according to the rules of the current locale.

The members of the structure with type `char *` are pointers to strings, any of which (except `decimal_point`) can point to a NULL value to indicate that the value is not available in the current locale or is of zero length. The members with type `char` are non-negative numbers, any of which can be `CHAR_MAX` to indicate that the value is not available in the current locale. The members include the following:

`char *decimal_point`

The decimal point character used to format non-monetary quantities.

`char *thousands_sep`

The character used to separate groups of digits before the decimal point character in formatted non-monetary quantities.

`char *grouping`

A string whose elements indicate the size of each group of digits in formatted, non-monetary quantities. The string may contain the following:

`CHAR_MAX`	No further grouping is to be performed.
`0`	The previous element is to be repeatedly used for the remaining digits.

`other`	The integer value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits before the current group.

`char *int_curr_symbol`

The international currency symbol that applies to the current locale. The first three characters contain the alphabetic international currency symbol. The fourth character (immediately preceding the null character) is the character used to separate the international currency symbol from the monetary quantity.

`char *currency_symbol`

The local currency symbol applicable to the current locale.

`char *mon_decimal_point`

The decimal point used to format monetary quantities.

`char *mon_thousands_sep`

The separator for groups of digits before the decimal point in formatted monetary quantities.

`char *mon_grouping`

A string whose elements indicate the size of each group of digits in formatted monetary quantities. The string may contain the following:

`CHAR_MAX`	No further grouping is to be performed.
`0`	The previous element is to be repeatedly used for the remaining digits.
`other`	The integer value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits before the current group.

`char *positive_sign`

The string used to indicate a non-negative-valued formatted monetary quantity.

`char *negative_sign`

The string used to indicate a negative-valued formatted monetary quantity.

`char int_frac_digits`

The number of fractional digits (those after the decimal point) to display in an internationally formatted monetary quantity.

`char p_cs_precedes`

Set to one or zero if the `currency_symbol` respectively precedes or succeeds the value for a non-negative formatted monetary quantity.

`char p_sep_by_space`

Set to one or zero if the `currency_symbol` respectively is or is not separated by a space from the value for a non-negative formatted monetary quantity.

`char n_cs_precedes`

Set to one or zero if the `currency_symbol` respectively precedes or succeeds the value for a negative formatted monetary quantity.

`char n_sep_by_space`

Set to one or zero if the `currency_symbol` respectively is or is not separated by a space from the value for a negative formatted monetary quantity.

`char p_sign_posn`

Set to a value indicating the positioning of the `positive_sign` for a non-negative formatted monetary quantity. The value may be:

`0`	Parentheses surround the quantity and currency symbol.
`1`	The sign string precedes the quantity and currency symbol.
`2`	The sign string succeeds the quantity and currency symbol.
`3`	The sign string immediately precedes the currency symbol.
`4`	The sign string immediately succeeds the currency symbol.

`char n_sign_posn`

Set to a value indicating the positioning of the `negative_sign` for a negative formatted monetary quantity. The values are the same as for `p_sign_posn`.

`localeconv()` returns a pointer to the filled in object. You cannot modify the structure pointed to by the return value, but a subsequent call to `localeconv()` can overwrite the structure. In addition, calls to `setlocale()` with categories `LC_ALL`, `LC_MONETARY`, or `LC_NUMERIC` may overwrite the contents of the structure.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

localtime()

Convert Calendar Time to Local Time

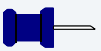
Syntax

```
#include <time.h>
struct tm *localtime(time_t *timer);
```

Description

`localtime()` converts a calendar time into a broken down time, expressed as local time.

`timer` is a pointer to the calendar time.



Note

`localtime()` returns a pointer to a static area which may be overwritten. To ensure data integrity, use the value or save it immediately.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`mktime()` (for more information on the time structure tm) `time()`

log()

Natural Logarithm

Syntax

```
#include <math.h>
double log(double x);
```

Description

`log()` returns the natural logarithm of `x`. A domain error occurs, `EDOM` stored in `errno`, if `x` is negative. A range error, `ERANGE` stored in `errno`, occurs if `x` is zero. `log()` returns `HUGE_VAL` for both errors.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Errors

`EDOM`
`ERANGE`

log10()

Base-Ten Logarithm

Syntax

```
#include <math.h>
double log10(double x);
```

Description

`log10()` returns the base-ten logarithm of `x`. A domain error, `EDOM` stored in `errno`, occurs if `x` is negative. A range error, `ERANGE` stored in `errno`, occurs if `x` is zero. `log10()` returns `HUGE_VAL` for both errors.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Errors

`EDOM`
`ERANGE`

longjmp()

Non-Local Goto

Syntax

```
#include <setjmp.h>
void longjmp(
     jmp_buf      env,
     int          val);
```

Description

`longjmp()` and `setjmp()` allow program control to return directly to a higher level function. They are most useful when dealing with errors and signals encountered in a low-level routine.

`env` is the program environment buffer.

`val` is the error status value.

`longjmp()` restores the environment saved by the most recent call to `setjmp()` in the same execution of the program, with the corresponding `jmp_buf` parameter. If there has been no such call or if the function containing the `setjmp()` call has returned in the interim, the behavior is undefined.

All accessible objects have values as of the time `longjmp()` was called, except that the value of objects of automatic storage duration that are local to the function containing the call to the corresponding `setjmp()` call and `longjmp()` call are indeterminate.

Because it bypasses the usual function call and return mechanisms, `longjmp()` executes correctly in contexts of interrupts, signals, and any of their associated functions. However, if `longjmp()` is called from a nested signal handler (that is, from a function called as a result of a signal raised during the handling of another signal), the behavior is undefined.

After `longjmp()` is completed, program execution continues as if the corresponding call to `setjmp()` had just returned the value specified by `val`. `longjmp()` cannot cause `setjmp()` to return the value zero. If `val` is zero, `setjmp()` returns the value `1`.

WARNING

Do not call `longjmp()` before `env` is initialized by `setjmp()` or if the function calling `setjmp()` has already returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`setjmp()`

_lrealloc()

Resize Block of Memory (Low-Overhead)

Syntax

```
#include <stdlib.h>
void *_lrealloc(
     void         *oldptr,
     unsigned long newsize,
     unsigned long oldsize);
```

Description

`_lrealloc()` resizes a block of memory.

`oldptr` is a pointer to the memory block. `oldptr` should be a value returned by a previous call to `_lmalloc()`, `_lcalloc()`, or `_lrealloc()`.

`newsize` specifies the size to use for the memory block.

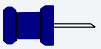
`oldsize` is the size of the old memory block.

`_lrealloc()` returns a pointer to a new memory block of size `newsize`. The pointer is aligned to store data of any type.

If `newsize` is smaller than `oldsize`, the contents of the old block are truncated and placed in the new block. Otherwise, all of the old block's contents begin the new block.

The results of `_lrealloc(NULL,10,0)` and `_lmalloc(10)` are the same.

`_lrealloc()` returns NULL if the requested memory is not available or if `newsize` is specified as zero.



Note

Use of the low-overhead allocation functions (`_lcalloc()`, `_lmalloc()`, and `_lrealloc()`) instead of the general allocation functions (`calloc()`, `malloc()`, and `realloc()`) saves 8 bytes per allocation because the low-overhead functions do not save the allocation size or the 4-byte check value. The low-overhead functions require that you keep track of the sizes of allocated spaces in memory.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`clib.l`

See Also

`calloc()`
`free()`
`_freemin()`
`_lcalloc()`
`_lfree()`
`_lmalloc()`
`malloc()`
`_mallocmin()`
`_os_srqmem()`
`_os_srtmem()`
`realloc()`

Syntax

```
#include <modes.h>
long lseek(
     int          path,
     long         position,
     int          place);
```

Description

`lseek()` repositions the file pointer for the file open on `path` to the byte offset given in `position`.

`path` is the path number of the open file.

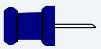
`position` is the new file position offset.

`place` determines from which file position the offset is based:

- `0` — From the beginning of the file.
- `1` — From the current position.
- `2` — From the end of the file.

Seeking to a location beyond end-of-file for a file open for writing and then writing to it creates a hole in the file. The hole contains data with no particular value, usually garbage remaining on the disk.

The returned value is the resulting position in the file. If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

On OS-9, the file pointer is a full unsigned 32-bits.

Do not use `lseek()` on a buffered file because the buffering routines use `fseek()` to keep track of the file pointer.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`fseek()`
`_os_seek()`

`I$Seek`	***OS-9 for 68K Technical Manual***
`I_SEEK`	***OS-9 Technical Manual***

makdir()

Create Directory

Syntax

```
#include <modes.h>
int makdir(
     char         *name,
     int          mode,
     int          perm[,
     int          size]);
```

Description

`makdir()` creates a directory file.

`name` is a pointer to the name of the directory. `mode` specifies the file access mode. Only the lower 16 bits of `mode` are used. `perm` specifies the file attribute permissions. Only the lower 16 bits of `perm` are used. If the `FAM_SIZE` (or `S_ISIZE`) bit is set in `mode`, the initial size of the directory is set to `size`.

If successful, `makdir()` returns zero. Otherwise, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_makdir()`
`I$MakDir` — ***OS-9 for 68K Technical Manual***
`I_MAKDIR` — ***OS-9 Technical Manual***

make_gdesc()

Make Global Descriptor Table Entry

Syntax

```
#include <regs.h>
#include <types.h>
void *make_gdesc(
     void         *tbl_entry_addr,
     u_int32      offset,
     u_int16      segment_attr);
```

Description

`make_gdesc()` makes an entry in the global descriptor table. It returns a pointer to the next entry.

`tbl_entry_addr` is a pointer to the address of the local descriptor table or the global descriptor table.

`offset` is the offset into the global descriptor table.

`segment_attr` is the segment attributes.

Note

`make_gdesc()` is only supported by the 80x86 family of processors version of the compiler.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

make_idesc()

Make Interrupt Descriptor Table Entry

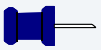
Syntax

```
#include <regs.h>
#include <types.h>

void *make_idesc(
     void         *tbl_entry_addr,
     u_int32      offset,
     u_int16      segment_attr,
     u_int16      selector);
```

Description

`make_idesc()` makes an entry in the interrupt descriptor table. It returns a pointer to the next entry. `tbl_entry_addr` is a pointer to the interrupt descriptor table. `offset` is an offset into the table. `segment_attr` specifies the segment attributes. `selector` specifies the selector entry value.



Note

`make_idesc()` is only supported by the 80x86 family of processors version of the compiler.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

Syntax

```
#include <memory.h>
#include <module.h>
mh_com *make_module(
     char         *name,
     int          size,
     int          attr,
     int          perm,
     int          typelang,
     int          color);
```

Description

`make_module()` creates a memory module with the specified name and attributes in the specified memory. `make_module()` is similar to `_mkdata_module()`. However, `make_module()` allows you to specify the type/language of the module and color of memory in which to make the module.

`mod_exec` is defined in the `module.h` header file.

`name` is a pointer to the name of the module.

`size` specifies the desired memory size of the module in bytes. The size value does not include the module header and CRC bytes. `size` is the amount of memory available for actual use. The memory in the module is initially cleared to zeroes.

`attr` specifies the module's attribute and revision level. Only the lower 16 bits of `attr` are used.

`perm` specifies the module permissions. Only the lower 16 bits of `perm` are used.

`typelang` specifies the type and language of the module. Only the lower 16 bits of `typelang` are used.

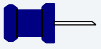
color indicates the specific memory type in which to load the module. The memory.h header file contains definitions of the memory types that you can specify:

Table 2-15 make_module() Color Memory Types

Memory Type	Description
SYSRAM or MEM_SYS	System RAM memory
VIDEO1	Video memory for plane A
VIDEO2	Video memory for plane B
MEM_ANY	No specific memory type

If color is MEM_ANY, the module is made in whatever memory the system allocates.

make_module() returns a pointer to the beginning of the module header. If the module cannot be created, -1 is returned and the appropriate error code is placed in the global variable errno.



Note

The name of the module created by make_module() always begins at offset $34 within the module. This implies that program modules, trap handlers, file managers, device drivers, and device descriptors cannot be made conveniently by this call.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

_os_datmod()

F$DatMod service request *OS-9 for 68K Technical Manual*

malloc()

Allocate Memory from Arena

Syntax

```
#include <stdlib.h>
void *malloc(size_t size);
```

Description

`malloc()` allocates space for an object. The value of the object is indeterminate.

`size` is the size of the memory block to allocate. If `size` is zero, NULL is returned.

`malloc()` returns either a null pointer (if sufficient free memory is not available) or a pointer to the allocated space.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`calloc()`
`free()`
`_os_srqmem()`
`_os9_srqmem()`
`realloc()`

`F$SRqMem`	***OS-9 for 68K Technical Manual***
`F_SRQMEM`	***OS-9 Technical Manual***

_mallocmin()

Set Minimum Allocation Size

Syntax

```
#include <stdlib.h>
void _mallocmin(unsigned size);
```

Description

`_mallocmin()` sets the minimum amount of memory that allocation functions may request from the system.

`size` is the minimum allocation size in bytes. `size` cannot be less than the system memory block size. Size must be greater than or equal to the larger of the system memory block size or 4K. If a smaller size is requested, `size` is automatically set to the system memory block size.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`clib.l`

See Also

`calloc()`
`_lcalloc()`
`_lmalloc()`
`_lrealloc()`
`malloc()`
`realloc()`

mblen()

Determine Number of Bytes in Multibyte Character

Syntax

```
#include <stdlib.h>
int mblen(
     const char   *str,
     size_t       num);
```

Description

`mblen()` determines the number of bytes contained in the multibyte character pointed to by `str`, if `str` is a not a null pointer. `mblen()` is similar to the following; however, the shift state of `mbtowc()` is unaffected:

```
mbtowc((wchar_t *)0, s, n);
```

`str` is a pointer to a multibyte character.

`num` is the number of bytes to search.

If `str` is a null pointer, `mblen()` returns a non-zero value (if the multibyte character encodings have state-dependent encodings) or returns a zero value.

If `str` is not a null pointer, `mblen()` returns:

- `0` if `str` points to the null character.
- The number of bytes contained in the multibyte character if the next `num` or fewer bytes form a valid multibyte character.
- `-1` if they do not form a valid multibyte character.

`mblen()` is supported for both the `C` and `JAPAN` locales.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library
clib.l

See Also
mbtowc()

mbstowcs()

Convert Sequence of Multibyte Characters

Syntax

```
#include <stdlib.h>

size_t mbstowcs(
     wchar_t      *pwcs,
     const char   *str,
     size_t       num);
```

Description

`mbstowcs()` converts a sequence of multibyte characters that begins in the initial shift state into a sequence of corresponding codes and stores not more than `num` codes in an array. No multibyte characters that follow a null character are examined or converted. A null character is converted into a code with value zero. Each multibyte character is converted as if by a call to `mbtowc()`, except that the shift state of `mbtowc()` is unaffected.

`pwcs` is a pointer to the array into which to store the converted values.

`str` is a pointer to the original sequence of multibyte characters.

`num` specifies the maximum number of elements to modify.

No more than `num` elements are modified in `pwcs`. If copying takes place between overlapping objects, the behavior is undefined.

If an invalid multibyte character is encountered, `mbstowc()` returns `(size_t)-1`. Otherwise, `mbstowcs()` returns the number of array elements modified, not including a terminating zero code, if any.

`mbstowcs()` is supported for both the `C` and `JAPAN` locales.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`wcstombs()`

Syntax

```
#include <stdlib.h>
int mbtowc(
     wchar_t      *pwc,
     const char   *mbchar,
     size_t       maxnum);
```

Description

`mbtowc()` determines the number of bytes contained in the multibyte character pointed to by `mbchar`, if `mbchar` is not a null pointer. It then determines the code for the value of type `wchar_t` that corresponds to that multibyte character. The value of the code corresponding to the null character is zero. If the multibyte character is valid and `pwc` is not a null pointer, `mbtowc()` stores the code in `pwc`. At most `maxnum` bytes of `mbchar` are examined.

`pwc` is a pointer to the array in which to store the code.

`mbchar` is a pointer to a multibyte character.

If `mbchar` is a null pointer, `mbtowc()` returns a non-zero value if multibyte character encodings have state-dependent encodings or returns a zero value.

If `mbchar` is not a null pointer, `mbtowc()` either returns:

- `0` if `mbchar` is a pointer to the null character.
- The number of bytes contained in the converted multibyte character if the next `maxnum` or fewer bytes form a valid multibyte character.
- `-1` if they do not form a valid multibyte character.

`maxnum` specifies the maximum number of bytes to examine.

The returned value is never greater than `maxnum` or `MB_CUR_MAX`.

`mbtowcs()` is supported for both the `C` and `JAPAN` locales.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`wctomb()`

memchr()

Memory Search

Syntax

```
#include <string.h>
void *memchr(
     const void   *mem_area,
     int          chr,
     size_t       size);
```

Description

`memchr()` locates the first occurrence of `chr`, converted to an `unsigned char`, in the initial `size` characters, each interpreted as `unsigned char`, of `mem_area`.

`mem_area` is a pointer to the memory to search.

`chr` is the character for which to search.

`size` is the size of the memory area to search.

`memchr()` returns a pointer to the located character, or a null pointer if the character does not occur in the object.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

memcmp()

Compare Memory

Syntax

```
#include <string.h>
int memcmp(
     const void   *string1,
     const void   *string2,
     size_t       size);
```

Description

`memcmp()` compares the first `size` characters of `string1` to the first `size` characters of `string2`.

`string1` is a pointer to a string to compare.

`string2` is a pointer to a string to compare.

`size` is the number of characters to compare.

`memcmp()` returns an integer that is:

- Positive, if `string1` is greater than `string2`.
- Zero, if `string1` equals `string2`.
- Negative, if `string1` is less than `string2`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

memcpy()

Copy Memory

Syntax

```
#include <string.h>
void *memcpy(
     void         *dest,
     const void   *source,
     size_t       num);
```

Description

`memcpy()` copies characters from one location to another. If copying takes place between overlapping objects, the behavior is undefined. `memcpy()` returns the value of `dest`.

`dest` is a pointer to the destination in memory.

`source` is a pointer to the memory to copy into `dest`.

`num` is the number of characters to copy.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <string.h>
void *memmove(
     void         *dest,
     const void   *source,
     size_t       num);
```

Description

`memmove()` copies characters from one location to another. Copying takes place as if the characters from the original location are first copied into a temporary array of `num` characters that does not overlap `dest` and `source`, and then the `num` characters from the temporary array are copied into `dest`.

`dest` is a pointer to the destination in memory.

`source` is a pointer to the memory to copy into `dest`.

`num` is the number of characters to copy.

`memmove()` returns the value of `dest`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

memset()

Fill Memory

Syntax

```
#include <string.h>
void *memset(
     void          *mem,
     int           fill,
     size_t        size);
```

Description

`memset()` copies the value of `fill`, converted to an `unsigned char`, into each of the first `size` characters of `mem`. `memset()` returns the value of `mem`.

`mem` is a pointer to the memory to fill.

`fill` is the value with which to fill an area of memory.

`size` is the number of characters to fill with `fill`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

_mkdata_module()

Create Data Memory Module

Syntax

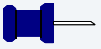
```
#include <module.h>
char *_mkdata_module(
     char         *name,
     unsigned     size,
     int          attr,
     int          perm);
```

Description

`_mkdata_module()` creates a data memory module. Other processes on the system can then access the data module by `modlink()`. The memory in the data module is initially cleared to zeroes.

`name` is a pointer to the desired module name.

`size` specifies the size of the module in bytes.



Note

The size value does not include the module header and CRC bytes. The specified size is the amount of memory available for actual data storage.

`attr` specifies the module's attribute and revision level. Only the lower 16 bits of `attr` are used.

`perm` specifies the module access permissions. Only the lower 16 bits of `perm` are used.

`_mkdata_module()` returns a pointer to the beginning of the module header.

If the data module cannot be created, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_datmod()`	
`F$DatMod`	***OS-9 for 68K Technical Manual***
`F_DATMOD`	***OS-9 Technical Manual***

mknod()

Create Directory

Syntax

```
#include <modes.h>
int mknod(
     char         *name,
     int          perm);
```

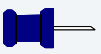
Description

`mknod()` creates a directory file.

`name` is a pointer to the name of the directory.

`perm` specifies the access permissions for the directory. Only the lower 16 bits of `perm` are used.

`mknod()` returns zero if the directory was successfully created. If the creation failed, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

`mknod()` does not make UNIX-style special files as there are no such files on OS-9 for 68K and OS-9. This is a historical function and is likely to be removed in a future release. Use `makdir()` in all new code.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

makdir()
_os_makdir()
I$MakDir — *OS-9 for 68K Technical Manual*
I_MAKDIR — *OS-9 Technical Manual*

mktemp()

Create Unique File Name

Syntax

```
#include <stdio.h>
char *mktemp(char *name);
```

Description

`mktemp()` ensures that the name of a temporary file is unique in the system and does not clash with any other file name.

`name` is a pointer to the template string.

A pointer to a template string is passed to `mktemp()`. The template string should look like a filename with six trailing `X`'s. `mktemp()` replaces the `X`'s with a letter and the current process ID. The letter is chosen so that the resulting name does not conflict with an existing file.

`mktemp()` returns a pointer to the file name or `NULL` if more than 26 `mktemp()` files exist.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`tmpnam()`
`tmpfile()`

mktime()

Convert Broken-Down Time to Calendar Time

Syntax

```
#include <time.h>
time_t mktime(struct tm *tp);
```

Description

`mktime()` converts the broken-down time, expressed as local time, into a calendar time value with the same encoding as the values returned by `time()`. The original values of the `tm_wday` and `tm_yday` fields of the structure are ignored, and the original values of the other fields are not restricted to the indicated ranges.

`tp` is a pointer to the broken-down time structure.

On successful completion, the `tm_wday` and `tm_yday` fields of the structure are set appropriately. The other fields are set to represent the specified calendar time, but with their values forced to the indicated ranges. The final value of `tm_mday` is not set until `tm_mon` and `tm_year` are determined.

`mktime()` returns the specified calendar time encoded as a value of type `time_t`. If the calendar time cannot be represented, `mktime()` returns the value `(time_t)-1`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

asctime()
getenv()
gmtime()
localtime()
time()

modcload()

Load Module into Colored Memory

Syntax

```
#include <memory.h>
#include <module.h>
mh_com *modcload(
     char         *modname,
     int          mode,
     int          memtype);
```

Description

`modcload()` opens the file specified by the pathlist. It reads one or more memory modules from the file into memory until an error or end of file is reached. `modname` is considered a pathlist and all modules in the specified file are loaded. A link is made to the first module loaded from the file and a pointer to it is returned.

`modname` is a pointer to the path to the module.

`mode` is the mode which opens the file to load. If any access `mode` is acceptable, specify zero for `mode`.

`memtype` indicates the specific memory type in which to load the module. The `memory.h` header file contains definitions of the memory types that may be specified:

Table 2-16 modcload() Memory Types

Memory Type	Description
`SYSRAM or MEM_SYS`	System RAM memory
`VIDEO1`	Video memory for plane A
`VIDEO2`	Video memory for plane B.
`MEM_ANY`	No specific memory type.

If `memtype` is `MEM_ANY`, the module may be loaded in whatever memory the system allocates.

If the load is successful, `modcload()` returns a pointer to the module. If the load fails, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

modf()

Return Parts of Real Number

Syntax

```
#include <math.h>
double modf(
     double       value,
     double       *num);
```

Description

`modf()` breaks `value` into integral and fractional parts, each of which has the same sign as the parameter. It stores the integral part as a double in `num`.

`value` is a real number.

`num` is a pointer to the integer portion of `value`.

`modf()` returns the signed fractional part of `value`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <module.h>
mh_com *modlink(
     char         *modname,
     int          typelang);
```

Description

`modlink()` searches the module directory for a module with the same name as that pointed to by `modname` and links to it, provided that `typelang` matches the respective value of Type/Language in the module. If the module is found, the module's link count is incremented by one.

`modname` is a pointer to the name of the module.

`typelang` is the type and language value of the module. If any module type or language is acceptable, specify zero for `typelang`. Only the lower 16 bits of `typelang` are used.

The `module.h` header file contains a structure appropriate for accessing the elements of system-defined memory modules.

If the link is successful, `modlink()` returns a pointer to the module. If the link fails, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

_os_link()	
_os_load()	
F$Link	*OS-9 for 68K Technical Manual*
F_LINKM	*OS-9 Technical Manual*

modload()

Load and Link to Memory Module

Syntax

```
#include <module.h>
mh_com *modload(
     char         *modname,
     int          accessmode);
```

Description

`modload()` loads all modules in the file specified by the path `modname`.

`modname` is a pointer to the name of the module.

`accessmode` specifies the module's access mode. If any module access mode is acceptable, specify zero for `accessmode`. Only the lower 16 bits of `accessmode` are used.

A link is made to the first module loaded from the file and a pointer to it is returned.

The `module.h` header file contains a structure appropriate for accessing the elements of system-defined memory modules.

If the load is successful, `modload()` returns a pointer to the module. If the load fails, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

To load a module using the shell `PATH` variable list, see `_os_loadp()` or `modloadp()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

_os_link()	
_os_load()	
_os_loadp()	
F$Load	***OS-9 for 68K Technical Manual***
F_LOAD	***OS-9 Technical Manual***

modloadp()

Load and Link to Memory Module Using PATH

Syntax

```
#include <module.h>
mh_com *modloadp(
     char         *modname,
     int          accessmode,
     char         *namebuffer);
```

Description

`modloadp()` loads all modules in the file specified by the path `modname`. The `PATH` environment variable determines alternate directories to search for the named module.

`PATH` can contain a list of directories to search if the program is not found in the module directory or the execution directory. `PATH` is set to any number of directory pathlists separated by colons:

```
/h0/cmds:/d0/cmds:/n0/droid/h0/special_stuff/cmds
```

`modname` is a pointer to the name of the module.

`accessmode` is the access mode for the module. Only the lower 16 bits of `accessmode` are used.

`namebuffer` is a pointer to an array containing the pathlist of the successfully loaded module. Any errors other than `EOS_PNNF` leave the path list used for the load attempt intact. `namebuffer` can be NULL if access to the path string is not required.

`modloadp()` returns a pointer to the module if the load is successful. If the load is unsuccessful, `-1` is returned and the appropriate error code is placed in `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

_os_loadp()
modload()
_os_load()

mq_close

Close a Message Queue

Syntax

```
#include <mqueue.h>
int mq_close(mqd_t mqdes);
```

Description

`mq_close()` removes the association between the message queue descriptor, `mqdes`, and its message queue. The results of using this descriptor after successful return from `mq_close()`—and until the return of this descriptor from a subsequent `mq_open()`—are undefined.

If the process has successfully attached a notification request to the message queue via `mqdes`, the attachment is removed and the message queue is available for another process to attach for notification.

Upon successful completion `mq_close()` returns a value of 0. Otherwise, the function returns -1 and sets `errno` to indicate the error.

Errors

EBADF	The `mqdes` argument is not a valid message queue descriptor.
ENOSYS	`mq_close()` not supported.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No
Standards:	POSIX

Library

`mq.l`

See Also

mq_open
mq_unlink

mq_getattr

Get Message Queue Attributes

Syntax

```
#include <mqueue.h>
int mq_getattr(mqd_t mqdes, struct mq_attr *mqstat);
```

Description

The `mqdes` argument specifies a message queue descriptor. `mq_getattr()` gets status information and attributes of the message queue and the open message queue description associated with the message queue descriptor. The results are returned in the `mq_attr` structure referenced by the `mqstat` argument.

Upon return, the following members shall have the values associated with the open message queue description as set when the message queue was opened and as modified by subsequent `mq_setattr()` calls: `mq_flags`.

The following attributes of the message queue shall be returned as set at message queue creation: `mq_maxmsg`, `mq_msgsize`.

Upon return, the following members within the `mq_attr` structure referenced by the `mqstat` argument shall be set according to the current state of the message queue: `mq_curmsgs`—the number of messages currently on the queue.

OS-9 specifies the following user-defined flag for use with `mq_getattr()` and `mq_setattr()`: `_MQ_O_NOTIFY_IMMEDIATE`. When this flag is specified by the `mq_setattr()` function (`mq_flags` field of `mqstat` parameter) then calls to either `_mq_notify_write()` or `mq_notify()` will notify immediately if appropriate. (Message queue nonfull for `_mq_notify_write()`, and message queue non-empty for `mq_notify()`) This extension provides message queue notification consistency with other OS-9 services such the "send signal on data ready" concept.

Upon successful completion `mq_getattr()` returns 0. Otherwise, the function returns -1 and sets `errno` to indicate the error.

Errors

EBADF	The `mqdes` argument is not a valid message queue descriptor.
ENOSYS	`mq_getattr()` not supported.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No
Standards:	POSIX

Library

`mq.l`

See Also

`mq_open`
`mq_send`
`mq_setattr`

mq_notify

Notify Process That a Message is Available on a Queue

Syntax

```
#include <mqueue.h>
int mq_notify(mqd_t                    mqdes,
              const struct sigevent *notification);
```

Description

If the argument notification is not `NULL`, this function registers the calling process to be notified of message arrival at an empty message queue associated with `mqdes`, the specified message queue descriptor. The notification specified by the notification argument is sent to the process when the message queue transitions from empty to nonempty. At any time, only one process can be registered for notification by a message queue. If the calling process or any other process has already registered for notification of message arrival at the specified message queue, subsequent attempts to register for that message queue fail.

If notification is `NULL` and the process is currently registered for notification by the specified message queue, the existing registration is removed.

When the notification is sent to the registered process, its registration is removed. The message queue is then available for registration.

If a process has registered for notification of message arrival at a message queue, and some thread is blocked in `mq_receive()` waiting to receive a message when a message arrives at the queue, the arriving message satisfies the appropriate `mq_receive()`. The resulting behavior is as if the message queue remains empty, and no notification is sent.

Upon successful completion, `mq_notify()` returns a value of 0. Otherwise, the function returns -1 and sets `errno` to indicate the error.

Errors

`EBADF`	The `mqdes` argument is not a valid message queue descriptor.

EBUSY	A process is already registered for notification by the message queue.
ENOSYS	`mq_notify()` not supported.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No
Standards:	POSIX

Library

`mq.l`

See Also

`mq_open`
`mq_send`
`mq_setattr`
`_MQ_0_NOTIFY_IMMEDIATE`

_mq_notify_write

Notify Process That a Space in the Message Queue is Available

Syntax

```
#include <mqueue.h>
int _mq_notify_write(mqd_t mqdes, const struct
sigevent *notification);
```

Description

If the arguement notification is not NULL, this function registers the calling process to be notified of message removal at a full message queue associated with the specified message queue descriptor, `mqdes`. The notification specified by the notification arguement is sent to the process when the message queue transitions from full to nonfull. At any time, only one process may be registered for notification by a message queue. If the calling process or any other process has already registered for notification of the specified message queue, subsequent attempts to register for that message queue will fail.

If notification is NULL and the process is currently registered for notification by the specified message queue, the existing registration is removed. The message queue is then available for registration.

If a process has registered for notification of a message queue and some thread is blocked in `mq_send()` waiting to send a message when the message queue transitions to nonfull, the arriving message will satisfy the appropriate `mq_send()`. The resulting behavior is as if the message queue remains full, and no notification is sent.

Errors

EBADF	The `mqdes` argument is not a valid message queue descriptor.
EBUSY	A process is already registered for notification by the message queue.
ENOSYS	`mq_notify()` not supported.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No
Standards:	POSIX

Library

`mq.l`

See Also

`mq_open`
`mq_send`
`mq_setattr`
`_MQ_0_NOTIFY_IMMEDIATE`

mq_open

Open a Message Queue

Syntax

```
#include <mqueue.h>
mqd_t mq_open(const char *name, int oflag, ...);
```

Description

`mq_open()` establishes the connection between a process and a message queue with a message queue descriptor. It creates an open message queue description that refers to the message queue, and it creates a message queue descriptor that refers to that open message queue description. The name argument points to a string naming a message queue. It is unspecified whether the name appears in the file system and is visible to other functions that take pathnames as arguments. The name argument conforms to the construction rules for a pathname. If name begins with the slash character, then processes calling `mq_open()` with the same value of name shall refer to the same message queue object, as long as that name has not been removed. If name does not begin with the slash character, the effect is implementation defined. The interpretation of slash characters other than the leading slash character in name is implementation defined. If the name argument is not the name of an existing message queue and creation is not requested, `mq_open()` fails and returns an error.

The `oflag` argument requests the desired receive and/or send access to the message queue. The requested access permission to receive messages or send messages is granted if the calling process would be granted read or write access, respectively, to an equivalently protected file.

The value of `oflag` is the bitwise inclusive OR of values from the following list. Applications shall specify exactly one of the first three values (access modes) below in the value of `oflag`:

`O_RDONLY` — Open the message queue for receiving messages. The process can use the returned message queue descriptor with `mq_receive()`, but not `mq_send()`. A

message queue may be open multiple times in the same or different processes for receiving messages.

`O_WRONLY` Open the queue for sending messages. The process can use the returned message queue descriptor with `mq_send()` but not `mq_receive()`. A message queue may be open multiple times in the same or different processes for sending messages.

`O_RDWR` Open the queue for both receiving and sending messages. The process can use any of the functions allowed for `O_RDONLY` and `O_WRONLY`. A message queue may be open multiple times in the same or different processes for sending messages.

Any combination of the remaining flags may be specified in the value of `oflag`:

`O_CREAT` This option is used to create a message queue, and it requires two additional arguments: `mode`, which is of type `mode_t`, and `attr`, which is a pointer to a `mq_attr` structure. If the pathname, `name`, has already been used to create a message queue that still exists, then this flag has no effect, except as noted under `O_EXCL`. Otherwise, a message queue is created without any messages in it. The user ID of the message queue is set to the effective user ID of the process, and the group ID of the message queue is set to the effective group ID of the process. The "file permission bits" is set to the value of mode. When bits in mode other than file permission bits are set, the effect is implementation defined. If `attr` is `NULL`, the message queue is

	created with implementation-defined default message queue attributes. If `attr` is non-NULL and the calling process has the appropriate privilege on name, the message queue `mq_maxmsg` and `mq_msgsize` attributes are set to the values of the corresponding members in the `mq_attr` structure referred to by `attr`. If `attr` is non-NULL, but the calling process does not have the appropriate privilege on name, `mq_open()` fails and returns an error without creating the message queue.
`O_EXCL`	If `O_EXCL` and `O_CREAT` are set, `mq_open()` fails if the message queue name exists. The check for the existence of the message queue and the creation of the message queue if it does not exist is atomic with respect to other processes executing `mq_open()` naming the same name with `O_EXCL` and `O_CREAT` set. If `O_EXCL` is set and `O_CREAT` is not set, the result is undefined.
`O_NONBLOCK`	The setting of this flag is associated with the open message queue description and determines whether a `mq_send()` or `mq_receive()` shall wait for resources or messages that are not currently available, or fail with `errno` set to [`EAGAIN`]. See `mq_send()` and `mq_receive()` for details.

`mq_open()` does not add or remove messages from the queue.

Upon successful completion, `mq_open()` returns a message queue descriptor. Otherwise, the function returns (`mqd_t`) -1 and sets `errno` to indicate the error.

Errors

EACCES	The message queue exists and the permissions specified by `oflag` are denied, or the message queue does not exist and permission to create the message queue is denied.
EEXIST	`O_CREAT` and `O_EXCL` are set and the named message queue already exists.
EINTR	The `mq_open()` operation was interrupted by a signal.
EINVAL	The `mq_open()` operation is not supported for the given name. The implementation shall document under what circumstances this error may be returned. `O_CREAT` was specified in `oflag`, the value of `attr` is not NULL, and either `mq_maxmsg` or `mq_msgsize` was less than or equal to zero.
EMFILE	Too many message queue descriptors or file descriptors are currently in use by this process.
ENAMETOOLONG	The length of the name string exceeds `{PATH_MAX}`, or a pathname component is longer than `{NAME_MAX}` while `{_POSIX_NO_TRUNC}` is in effect.
ENFILE	Too many message queues are currently open in the system.
ENOENT	`O_CREAT` is not set and the named message queue does not exist.
ENOSPC	There is insufficient space for the creation of the new message queue.
ENOSYS	`mq_open()` not supported.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No
Standards:	POSIX

Library

`mq.l`

See Also

`mq_close`
`mq_receive`
`mq_send`
`mq_setattr`
`mq_getattr`
`mq_unlink`

mq_receive

Receive a Message From a Message Queue

Syntax

```
#include <mqueue.h>
size_t mq_receive(mqd_t        mqdes,
                  char         *msg_ptr,
                  size_t       msg_len,
                  unsigned int *msg_prio);
```

Description

`mq_receive()` is used to receive the oldest of the highest priority message(s) from the message queue specified by `mqdes`. If the size of the buffer in bytes, specified by the `msg_len` argument, is less than the `mq_msgsize` attribute of the message queue, the function fails and returns an error. Otherwise, the selected message is removed from the queue and copied to the buffer pointed to by the `msg_ptr` argument.

If the argument `msg_prio` is not NULL, the priority of the selected message is stored in the location referenced by `msg_prio`.

If the specified message queue is empty and `O_NONBLOCK` is not set in the message queue description associated with `mqdes`, `mq_receive()` blocks until a message is enqueued on the message queue or until `mq_receive()` is interrupted by a signal. If more than one thread is waiting to receive a message when a message arrives at an empty queue and the Process Scheduling option is supported, then the thread of highest priority that has been waiting the longest is selected to receive the message. Otherwise, it is unspecified which waiting thread receives the message. If the specified message queue is empty and `O_NONBLOCK` is set in the message queue description associated with `mqdes`, no message is removed from the queue, and `mq_receive()` returns an error.

Upon successful completion, `mq_receive()` returns the length of the selected message in bytes and the message is removed from the queue. Otherwise, no message is removed from the queue, the function returns a -1, and sets `errno` to indicate the error.

Errors

EAGAIN	O_NONBLOCK was set in the message description associated with mqdes, and the specified message queue is empty.
EBADF	The mqdes argument is not a valid message queue descriptor open for reading.
EMSGSIZE	The specified message buffer size, msg_len, is less than the message size attribute of the message queue.
EINTR	The mq_receive() operation was interrupted by a signal.
ENOSYS	The mq_receive() function is not supported by this implementation.
EBADMSG	A data corruption problem with the message has been detected.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No
Standards:	POSIX

Library

mq.l

See Also

mq_send

mq_send

Send a Message to a Message Queue

Syntax

```
#include <mqueue.h>
int mq_send(mqd_t          mqdes,
            const char     *msg_ptr,
            size_t         msg_len,
            unsigned int   msg_prio);
```

Description

`mq_send()` adds the message pointed to by the argument `msg_ptr` to the message queue specified by `mqdes`. The `msg_len` argument specifies the length of the message in bytes pointed to by `msg_ptr`. The value of `msg_len` is less than or equal to the `mq_msgsize` attribute of the message queue, otherwise `mq_send()` fails.

If the specified message queue is not full, `mq_send()` behaves as if the message is inserted into the message queue at the position indicated by the `msg_prio` argument. A message with a larger numeric value of `msg_prio` is inserted before messages with lower values of `msg_prio`. A message is inserted after other messages in the queue, if any, with equal `msg_prio`. The value of `msg_prio` is less than `{MQ_PRIO_MAX}`.

If the specified message queue is full and `O_NONBLOCK` is not set in the message queue description associated with `mqdes`, `mq_send()` blocks until space becomes available to enqueue the message, or until `mq_send()` is interrupted by a signal. If more than one thread is waiting to send when space becomes available in the message queue and the Process Scheduling option is supported, then the thread of the highest priority that has been waiting the longest is unblocked to send its message. Otherwise, it is unspecified which waiting thread is unblocked. If the specified message queue is full and `O_NONBLOCK` is set in the message queue description associated with `mqdes`, the message is not queued and `mq_send()` returns an error.

Upon successful completion, `mq_send()` returns a value of 0. Otherwise, no message is enqueued, the function returns -1, and `errno` is set to indicate the error.

Errors

`EAGAIN`	The `O_NONBLOCK` flag is set in the message queue description associated with `mqdes`, and the specified message queue is full.
`EBADF`	The `mqdes` argument is not a valid message queue descriptor open for writing.
`EINTR`	A signal interrupted the call to `mq_send()`.
`EINVAL`	The value of `msg_prio` was outside the valid range.
`EMSGSIZE`	The specified message length, `msg_len`, exceeds the message size attribute of the message queue.
`ENOSYS`	`mq_send()` not supported.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No
Standards:	POSIX

Library

`mq.l`

See Also

`mq_receive`
`mq_setattr`

mq_setattr

Set Message Queue Attributes

Syntax

```
#include <mqueue.h>
int mq_setattr(mqd_t           mqdes,
               const struct    mq_attr *mqstat,
               struct mq_attr  *omqstat);
```

Description

`mq_setattr()` is used to set attributes associated with the open message queue description referenced by the message queue descriptor specified by `mqdes`.

The message queue attributes corresponding to the following members defined in the `mq_attr` structure are set to the specified values upon successful completion of `mq_setattr()`:

`mq_flags`	The value of this member is the bitwise logical OR of zero or more of `O_NONBLOCK` and any implementation-defined flags.

The values of the `mq_maxmsg`, `mq_msgsize`, and `mq_curmsgs` members of the `mq_attr` structure are ignored by `mq_setattr()`.

If `omqstat` is non-NULL, the function `mq_setattr()` stores, in the location referenced by `omqstat`, the previous message queue attributes and the current queue status. These values are the same as would be returned by a call to `mq_getattr()` at that point.

OS-9 specifies the following user-defined flag for use with `mq_getattr()` and `mq_setattr()`: `_MQ_O_NOTIFY_IMMEDIATE`. When this flag is specified by the `mq_setattr()` function (`mq_flags` field of `mqstat` parameter) then calls to either `_mq_notify_write()` or `mq_notify()` will notify immediately if appropriate. (Message queue nonfull for `_mq_notify_write()`, and message queue non-empty for `mq_notify()`) This extension provides message queue notification consistency with other OS-9 services such the "send signal on data ready" concept.

Upon successful completion, `mq_setattr()` returns a value of 0 and the attributes of the message queue are changed as specified. Otherwise, the message queue attributes are unchanged, and the function returns a value of -1 and sets `errno` to indicate the error.

Errors

EBADF	The `mqdes` argument is not a valid message queue descriptor.
ENOSYS	`mq_setattr()` not supported.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No
Standards:	POSIX

Library

`mq.l`

See Also

`mq_open`
`mq_send`

mq_unlink

Remove a Message Queue

Syntax

```
#include <mqueue.h>
int mq_unlink(const char *name);
```

Description

`mq_unlink()` removes the message queue named by the pathname name. After a successful call to `mq_unlink()` with name, a call to `mq_open()` with name fails if the flag `O_CREAT` is not set in flags. If one or more processes have the message queue open when `mq_unlink()` is called, destruction of the message queue is postponed until all references to the message queue have been closed. Calls to `mq_open()` to re-create the message queue may fail until the message queue is actually removed. However, the `mq_unlink()` call need not block until all references have been closed; it may return immediately.

Upon successful completion, `mq_unlink()` returns a value of 0. Otherwise, the named message queue is not changed by this function call, and the function returns a value of -1 and sets `errno` to indicate the error.

Errors

`EACCES`	Permission is denied to unlink the named message queue.
`ENAMETOOLONG`	The length of the name string exceeds `{NAME_MAX}` while `{_POSIX_NO_TRUNC}` is in effect.
`ENOENT`	The named message queue does not exist.
`ENOSYS`	`mq_unlink()` not supported.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No
Standards:	POSIX

Library

`mq.l`

See Also

`mq_close`
`mq_open`

munlink()

Unlink from Module

Syntax

```
#include <module.h>
int munlink(mh_com *module);
```

Description

`munlink()` informs the system that the process no longer requires the specified module. The module's link count is decremented, and the module is removed from memory if the link count reaches zero. `module` is a pointer to the module. It must have been a pointer returned by `modcload()`, `modload()`, `modloadp()`, `modlink()`, `_os_link()`, `_os_load()`, or `_os_loadp()`. On error, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`modcload()`
`modlink()`
`modload()`
`modloadp()`
`_os_link()`
`_os_load()`
`_os_loadp()`
`_os_unlink()`

`F$Unlink`	***OS-9 for 68K Technical Manual***
`F_UNLINK`	***OS-9 Technical Manual***

munload()

Unload Module

Syntax

```
#include <module.h>
int munload(
     char         *name,
     int          typelang);
```

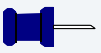
Description

`munload()` informs the system that the process no longer requires the module `name` which has a type/language of `typelang`. The module's link count is decremented. The module is removed from memory when the link count reaches zero.

`name` is a pointer to the name of the module.

`typelang` specifies the type and language of the module. If `typelang` is zero, any module may be unloaded. Only the lower 16 bits of `typelang` are used.

On error, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

`munload()` differs from `munlink()` in that it unlinks by module name rather than module pointer. Attempting to unlink a module by performing a link to a module by name to determine its address, then unlinking it twice does not work because the first unlink removes the module from the process' address space.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`munlink()`	
`_os_link()`	
`_os_load()`	
`_os_unload()`	
`F$UnLoad`	***OS-9 for 68K Technical Manual***
`F_UNLOAD`	***OS-9 Technical Manual***

offsetof()

Expand Value to Integral Constant Expression

Syntax

```
#include <stddef.h>
int offsetof(type, mbr_desig);
```

Description

`offsetof()` expands to an integral constant expression. The integral constant expression has a type of `size_t`, the value of which is the offset in bytes to the structure member from the beginning of the structure.

`type` specifies the type of the structure.

`mbr_desig` specifies the member name of the structure for which you want to compute the offset. If the member is a bit field, the behavior is undefined.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

open()

Open File

Syntax

```
#include <modes.h>
int open(
     char         *name,
     int          mode);
```

Description

`open()` opens an existing file.

`name` is a pointer to the name of the file.

`mode` specifies the access mode for the file. The values for `mode` are defined in the `modes.h` header file. Only the lower 16 bits of `mode` are used.

`open()` returns a path number identifying the file when I/O is performed. If the open fails, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`close()`
`creat()`
`_os_create()`
`_os_open()`
`I$Open` — ***OS-9 for 6K Technical Manual***
`_os_open()` — ***OS-9 Technical Manual***

opendir()

Open Directory

Syntax

```
#include <dir.h>
DIR *opendir(char *filename);
```

Description

`opendir()` opens the specified directory and associates a directory stream with it. `opendir()` returns a pointer identifying the directory stream in subsequent operations. The pointer `NULL` is returned if `filename` cannot be accessed or if it cannot allocate enough memory to hold the entire `DIR` structure.

`filename` is a pointer to the name of the directory.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

`closedir()`
`readdir()`
`rewinddir()`
`seekdir()`
`telldir()`

_os_acqlk()

Acquire Ownership of Resource Lock

Syntax

```
#include <lock.h>
#include <types.h>
error_code _os_acqlk(
     lk_desc      *lock,
     signal_code  *signal);
```

Description

`_os_acqlk()` acquires ownership of a resource lock (that is, it attempts to gain exclusive access to a resource).

`lock` is a pointer to the `lk_desc` structure of the lock that you are trying to acquire.

`signal is` a pointer to the location where `_os_acqlk()` stores the signal that prematurely terminated the acquisition of the lock, if applicable.

If the lock is not owned by another process, the calling process is granted ownership and the call returns without error.

If the lock is already owned, the calling process is suspended and inserted into a waiting queue for the resource based on relative scheduling priority.

When ownership of the lock is released, the next process in the queue is granted ownership and is activated. The activated process returns from the `_os_acqlk()` call without error. If a process received a signal while waiting on a lock, the process is activated without gaining ownership of the lock. The process returns from the `_os_acqlk()` call with an `EOS_SIGNAL` error code and the signal code returned in the `signal` pointer.

Note

If a waiting process receives an S_WAKEUP signal, the signal code does not register and is zero.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`lock.l`

See Also

F_ACQLK	***OS-9 Technical Manual***
F_CAQLK	***OS-9 Technical Manual***
F_CRLK	***OS-9 Technical Manual***
F_DDLK	***OS-9 Technical Manual***
F_DELLK	***OS-9 Technical Manual***
F_RELLK	***OS-9 Technical Manual***
F_WAITLK	***OS-9 Technical Manual***

_os9_alarm_atdate()

Send Signal at Gregorian Date/Time

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os9_alarm_atdate(
     alarm_id     *alrm_id,
     signal_code  signal,
     u_int32      time,
     u_int32      date);
```

OS-9 Attributes

Compatibility	State	Threads Compatibility	Reentrant
OS-9 for 68K	User	Safe	Yes

Description

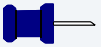
`_os9_alarm_atdate()` sends a signal to the caller at a specific date and time. The alarm signal is sent anytime the system date/time becomes greater than or equal to the alarm time.

`alrm_id` is a pointer to the location where `_os9_alarm_atdate()` stores the alarm ID.

`signal` specifies the signal code.

`date` is in the format `yyyymmdd`.

`time` is in the format `00hhmmss`.



Note

`_os9_alarm_atdate()` only allows you to specify the time to the nearest second. However, it does adjust if the system's date and time have changed (using `_os9_setime()`).

Attributes

Operating System:	OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Alarm` ***OS-9 for 68K Technical Manual***

_os_alarm_atime()

Send Signal at Specified Time

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os_alarm_atime(
     alarm_id     *alrm_id,
     signal_code  signal,
     u_int32      time);
```

Description

`_os_alarm_atime()` sends a signal when the system time reaches or passes the specified time. The time value is considered to be an absolute value in seconds since 1 January 1970 Greenwich Mean Time. `alrm_id` is a pointer to the location where `_os_alarm_atime()` stores the alarm ID. `signal` is the signal code of the signal to send. `time` specifies the time.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_alarm_set()`	
`_os9_alarm_atdate()`	
`_os9_alarm_atjul()`	
`F$Alarm`	***OS-9 for 68K Technical Manual***
`F_ALARM`	***OS-9 Technical Manual***

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os9_alarm_atjul(
     alarm_id     *alrm_id,
     signal_code  signal,
     u_int32      time,
     u_int32      date);
```

OS-9 Attributes

Compatibility	State	Threads Compatibility	Re-entrant
OS-9 for 68K	User	Safe	Yes

Description

`_os9_alarm_atjul()` sends a signal to the caller at a specific Julian date and time. The alarm signal is sent anytime the system date/time becomes greater than or equal to the alarm time.

`alrm_id` is a pointer to the location where `_os9_alarm_atjul()` stores the alarm ID.

`signal` specifies the signal code.

`time` is the seconds after midnight.

`date` is the Julian day number.

Note

`_os9_alarm_atjul()` only allows you to specify the time to the nearest second. However, it does adjust if the system's date and time have changed (using `_os9_setime()`).

Attributes

Operating System:	OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Alarm` ***OS-9 for 68K Technical Manual***

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os_alarm_cycle(
     alarm_id     *alrm_id,
     signal_code  signal,
     u_int32      time);
```

Description

`_os_alarm_cycle()` sends a signal after the specified time interval has elapsed and then resets the alarm. This provides a recurring periodic signal. The time interval may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second. The minimum time interval allowed is 2 system clock ticks. `alrm_id` is a pointer to the location where `_os_alarm_cycle()` stores the alarm ID. `signal` is the signal code of the signal to send. `time` specifies the time interval between signals.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_alarm_set()`	
`F$Alarm`	***OS-9 for 68K Technical Manual***
`F_ALARM, A_CYCLE`	***OS-9 Technical Manual***

_os_alarm_delete()

Remove Pending Alarm Request

Syntax

```
#include <alarm.h>
#include <types.h>

error_code _os_alarm_delete(
     alarm_id     alrm_id);
```

Description

`_os_alarm_delete()` removes a cyclic alarm or any alarm that has not expired.

`alrm_id` specifies the alarm to remove. If `alrm_id` is zero, all pending alarm requests for the owner are removed.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Alarm`	***OS-9 for 68K Technical Manual***
`F_ALARM, F_DELET`	***OS-9 Technical Manual***

Syntax

```
#include <alarm.h>
#include <types.h>

error_code _os_alarm_reset(
     alarm_id     alrm_id,
     signal_code  signal,
     u_int32      time);
```

Description

`_os_alarm_reset()` sets an existing alarm to send after the specified time interval has elapsed.

`alrm_id` is the alarm ID to reset.

`signal` is the signal code of the signal to send.

`time` may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second. Usually, the minimum time interval allowed is 2 clock ticks.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_alarm_set()`	
`F$Alarm`	***OS-9 for 68K Technical Manual***
`F_ALARM, A_RESET`	***OS-9 Technical Manual***

_os_alarm_set()

Send Signal After Specified Time Interval

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os_alarm_set(
     alarm_id     *alrm_id,
     signal_code  signal,
     u_int32      time);
```

Description

`_os_alarm_set()` sends one signal after the specified time interval has elapsed.

`alrm_id` is a pointer to the location where `_os_alarm_set()` stores the alarm ID.

`signal` is the signal code of the signal to send.

`time` may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second. Usually, the minimum time interval allowed is 2 system clock ticks.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Alarm`	***OS-9 for 68K Technical Manual***
`F_ALARM, A_SET`	***OS-9 Technical Manual***

_os_alias()

Create Device Alias

Syntax

```
#include <io.h>
#include <types.h>
error_code _os_alias(
     char         *alias_name,
     char         *real_name);
```

Description

`_os_alias()` creates an alternate name for a device pathlist. Processes can then reference a specific device pathlist with a shorter or more convenient name.

`alias_name` is a pointer to the alternate name.

`real_name` is a pointer to the actual device name; it must exist. OS-9 does not validate its existence.

WARNING

Do not use a real device name as `alias_name`.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`I_ALIAS` ***OS-9 Technical Manual***

_os9_allbit()

Set Bits in Bit Map

Syntax

```
#include <memory.h>
#include <types.h>
error_code _os9_allbit(
     u_int32      bit_number,
     u_int32      count,
     void         *address);
```

Description

`_os9_allbit()` sets bits in a buffer that represents a bit map. Bit numbers range from `0` to `n - 1`, where `n` is the number of bits in the bit map.

`bit_number` specifies the bit number of the first bit to set. Only the lower 16 bits of `bit_number` are used.

`count` specifies the number of bits to set. Only the lower 16 bits of `count` are used.

`address` is a pointer to the base address of a bit map.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os9_schbit()`
`_os9_delbit()`
`F$AllBit` ***OS-9 for 68K Technical Manual***

_os9_allpd()

Allocate Fixed-Length Block of Memory

Syntax

```
#include <process.h>
#include <types.h>
error_code _os9_allpd(
     void         *table,
     u_int16      *number,
     void         **ptr);
```

Description

`_os9_allpd()` allocates fixed-length blocks of system memory. It allocates and initializes (to zeros) a block of storage and returns its address.

`table` is a pointer to the process/path table.

`number` is a pointer to the location where `_os9_allpd()` stores the process/path number.

`ptr` is a pointer to the location where `_os9_allpd()` stores the pointer to the process/path descriptor.

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os9_findpd()`
`_os9_retpd()`
`F$AllPD` ***OS-9 for 68K Technical Manual***

_os_alltsk()

Allocate Task

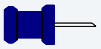
Syntax

```
#include <process.h>
#include <types.h>
error_code _os_alltsk(pr_desc *proc_desc);
```

Description

`_os_alltsk()` initializes the protection hardware for a newly activated process.

`proc_desc` is a pointer to the process descriptor.



Note

`_os_alltsk()` only performs a useful function on MMU/SSM systems. On non-MMU/SSM systems, `_os_alltsk()` simply returns without initializing the protection hardware.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$AllTsk`	***OS-9 for 68K Technical Manual***
`F_ALLTSK`	***OS-9 Technical Manual***

_os_alocproc()

Allocate Process Descriptor

Syntax

```
#include <process.h>
#include <types.h>

error_code _os_alocproc(
     process_id   *proc_id,
     pr_desc      **proc_desc);
```

Description

`_os_alocproc()` allocates and initializes a process descriptor. The descriptor's address is stored in the process descriptor table.

`proc_id` is a pointer to the location where `_os_alocproc()` stores the process ID.

`proc_desc` is a pointer to the location where `_os_alocproc()` stores the pointer to the process descriptor.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$AllPrc` ***OS-9 for 68K Technical Manual***

_os_altmdir()

Set Alternate Working Module Directory

Syntax

```
#include <moddir.h>
#include <types.h>
error_code _os_altmdir(char *name);
```

Description

`_os_altmdir()` establishes an alternate working module directory for a process.

`name` specifies the name of the module directory.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F_ALTMDIR`	***OS-9 Technical Manual***

_os_aproc()

Insert Process in Active Process Queue

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_aproc(process_id proc_id);
```

Description

`_os_aproc()` inserts the process specified by `proc_id` into the active process queue so that it may be scheduled for execution.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_nproc()`

`F$AProc`	***OS-9 for 68K Technical Manual***
`F_ALLPRC`	***OS-9 Technical Manual***

_os9_aproc()

Insert Process in Active Process Queue

Syntax

```
#include <process.h>
#include <types.h>

error_code _os9_aproc(pr_desc *proc_desc);
```

Description

`_os9_aproc()` inserts a process into the active process queue so that it may be scheduled for execution.

`proc_desc` is a pointer to the process descriptor.

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_nproc()`
F$AProc ***OS-9 for 68K Technical Manual***

Syntax

```
#include <io.h>
#include <modes.h>
#include <types.h>
error_code _os_attach(
     char         *name,
     u_int32      mode,
     dev_list     **dev_tbl);
```

Description

`_os_attach()` causes a new I/O device to become known to the system or verifies that the device is already attached.

`name` is a pointer to the I/O device's name.

`mode` is the access mode. It can be `FAM_READ`, `FAM_WRITE`, `S_IREAD`, or `S_IWRITE`. `mode` may be used to verify that subsequent read and/or write operations are permitted. Only the lower 16 bits of `mode` are used.

`dev_tbl` is a pointer to the location where `_os_attach()` stores the pointer to the device list entry.

If the descriptor `name` is found, `_os_attach()` links to the I/O device's file manager and device driver.

If the device was already attached, `_os_attach()` returns a pointer to the pointer to the device table entry containing the address of the I/O device's file manager and device driver in `dev_tbl`.

If the device is not already attached, `_os_attach()` creates a new device table entry and initializes the device. It returns a pointer to the pointer to the entry in `dev_tbl`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_detach()`

`I$Attach`	***OS-9 for 68K Technical Manual***
`I_ATTACH`	***OS-9 Technical Manual***

Syntax

For OS-9 for 68K:

```
#include <cache.h>* for OS-9 for  68K systems
#include <regs.h>
#include <types.h>

error_code _os_cache(u_int32 control);
```

For OS-9:

```
#include <regs.h>
#include <types.h>

error_code _os_cache(
     u_int32     control,
     void        *addr,
     u_int32     size);
```

Description

`_os_cache()` performs operations on the system instruction and/or data caches, if there are any.

If `control` is zero, the system instruction and data caches are flushed. Only non-super-group, user-state processes may perform this operation.

Any program that builds or changes executable code in memory must flush the instruction cache with `_os_cache()` before executing the new code. This is necessary because the hardware instruction cache is not updated by data (write) accesses and may contain the unchanged instruction(s). For example, if a subroutine builds a system call on its stack, you must execute the `_os_cache()` call to flush the instruction cache before executing the temporary instructions.

For More Information

Refer to your operating system technical manual for a description of the possible values for `control`.

`addr` specifies the target address for the flush operation.

`size` indicates the size of the target memory area to be flushed.

Note

The cache should be flushed before trying to disable the data cache.

Only super-group processes may perform precise operations of `F$CCtl`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

`_os_scache()`

`F$CCtl`	***OS-9 for 68K Technical Manual***
`F_CCTL`	***OS-9 Technical Manual***

Syntax

```
#include <lock.h>
#include <types.h>
error_code _os_caqlk(lk_desc *lock);
```

Description

`_os_caqlk()` is used to conditionally acquire ownership of a resource lock.

`lock` is a pointer to the lock.

If another process does not own the lock, the calling process is granted ownership and the call returns without error.

If the lock is already owned, the calling process is not suspended. Instead, it returns from the `_os_caqlk()` call with an `EOS_NOLOCK` error and is not granted ownership of the resource lock.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`lock.l`

See Also

`F_ACQLK`	***OS-9 Technical Manual***
`F_CAQLK`	***OS-9 Technical Manual***
`F_CRLK`	***OS-9 Technical Manual***
`F_DDLK`	***OS-9 Technical Manual***
`F_DELLK`	***OS-9 Technical Manual***
`F_RELLK`	***OS-9 Technical Manual***
`F_WAITLK`	***OS-9 Technical Manual***

_os_chain()

Execute New Primary Module

Syntax

```
#include <process.h>
#include <module.h>
#include <types.h>
error_code _os_chain(
     u_int32      priority,
     u_int32      path_cnt,
     char         *mod_name,
     void         *params,
     u_int32      param_size,
     u_int32      mem_size,
     u_int32      type_lang);
```

Description

`_os_chain()` executes a new program without the overhead of creating a new process. It is functionally similar to a Fork command followed by an Exit. `_os_chain()` effectively resets the calling process' program and data memory areas and begins executing a new primary module. Open paths are not closed or otherwise affected.

`priority` is the priority of the process. Only the lower 16 bits of `priority` are used.

`path_cnt` specifies the number of I/O paths to copy. Only the lower 16 bits of `path_cnt` are used.

`mod_name` is a pointer to the new program to execute.

`params` is a pointer to the parameter block.

`param_size` specifies the size of the parameter block.

`mem_size` specifies the additional memory size in bytes.

`type_lang` specifies the desired module type/language. `type_lang` must be either program/object (`MT_PROGRAM`, `ML_OBJECT`) or zero (for any). Only the lower 16 bits of `type_lang` are used.

Note

On OS-9 for 68K, `_os_chain()` never returns to the calling process. On OS-9, an error is returned only if there is not enough memory to hold the parameters. If an error occurs during the `chain`, it is returned as an exit status to the parent of the process performing the `chain`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_fork()`
`_os_load()`
`F$Chain` ***OS-9 for 68K Technical Manual***
`F_CHAIN` ***OS-9 Technical Manual***

_os_chainm()

Execute New Primary Module Given Pointer to Module

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_chainm(
     u_int32     priority,
     u_int32     path_cnt,
     mh_com      *mod_head,
     void        *params,
     u_int32     param_size,
     u_int32     mem_size);
```

Description

`_os_chainm()` executes a new program without the overhead of creating a new process. It is functionally similar to a Fork command followed by an Exit. `_os_chainm()` effectively resets the calling process' program and data memory areas and begins executing a new primary module. Open paths are not closed or otherwise affected.

`_os_chainm()` is similar to `_os_chain()`. However, it is passed a pointer to the module in memory to chain to instead of the module name.

`priority` is the priority of the process. Only the lower 16 bits of `priority` are used.

`path_cnt` is the number of I/O paths to copy. Only the lower 16 bits of `path_cnt` are used.

`mod_head` is a pointer to the module header.

`params` is a pointer to the parameter block.

`param_size` specifies the size of the parameter block.

`mem_size` specifies the additional memory size in bytes.

Note

An error is returned only if there is not enough memory to hold the parameters. If an error occurs during the `chainm`, it is returned as an exit status to the parent of the process performing the `chainm`.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F_CHAIN`	***OS-9 Technical Manual***
`F_CHAINM`	***OS-9 Technical Manual***
`F_EXIT`	***OS-9 Technical Manual***
`F_FORK`	***OS-9 Technical Manual***
`F_LOAD`	***OS-9 Technical Manual***

_os_chdir()

Change Working Directory

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_chdir(
     char         *name,
     u_int32      mode);
```

Description

`_os_chdir()` changes a process' working directory to the directory file specified by the pathlist. You may change the execution or the data directory (or both), depending on the access mode you specify. The specified file must be a directory file, and the caller must have access permission for the specified mode.

`name` is a pointer to the pathlist.

`mode` specifies the access mode. Only the lower 16 bits of `mode` are used.

For More Information

Refer to ***OS-9 Technical Manual*** or the ***OS-9 for 68K Technical Manual*** for the available modes.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`I$ChgDir`	***OS-9 for 68K Technical Manual***
`I_CHDIR`	***OS-9 Technical Manual***

_os_chkmem()

Check Memory Block's Accessibility

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_chkmem(
     u_int32      size,
     u_int32      mode,
     void         *mem_ptr,
     pr_desc      *proc_desc);
```

Description

`_os_chkmem()` determines if a process has access to a specified memory block.

`size` specifies the size of the memory area.

`mode` specifies the permissions to check. Only the lower 16 bits of `mode` are used.

`mem_ptr` is a pointer to the beginning of the memory to check.

`proc_desc` is a pointer to the process descriptor of the target process.

Note

`_os_chkmem()` only performs a useful function on MMU systems. On non-MMU systems, `_os_chkmem()` simply returns without checking the memory block's accessibility.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F$ChkMem	***OS-9 for 68K Technical Manual***
F_CHKMEM	***OS-9 Technical Manual***

_os_chmdir()

Change Process' Current Module Directory

Syntax

```
#include <moddir.h>
#include <types.h>
error_code _os_chmdir(char *name);
```

Description

`_os_chmdir()` changes a process' current module directory to the directory specified by `name`.

`name` is a pointer to a full pathlist or a pathlist relative to the current module directory. To change to the system's root module directory, specify a slash (`/`) for `name`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_close()

Close Path to File/Device

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_close(path_id path);
```

Description

`_os_close()` terminates the I/O path specified by `path`.

`path` is the path ID returned from a previous call to `_os_open()`, `_os_create()`, or `_os_dup()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_create()`
`_os_detach()`
`_os_dup()`
`_os_exit()`
`_os_open()`
`I$Close` — ***OS-9 for 68K Technical Manual***
`I_CLOSE` — ***OS-9 Technical Manual***

_os_clrsigs()

Clear Process Signal Queue

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_clrsigs(process_id proc_id);
```

Description

`_os_clrsigs()` removes any queued signals that have been sent to the target process.

`proc_id` specifies the target process ID number.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_sigmask()`

_os_cmdperm()

Change Permissions of Module Directory

Syntax

```
#include <moddir.h>
#include <types.h>
error_code _os_cmdperm(
     char         *name,
     u_int32      perm);
```

Description

`_os_cmdperm()` changes the access permissions of an existing module directory. This makes it possible to restrict access to a particular module directory.

`name` is a pointer to the name of the existing module directory.

`perm` specifies the new permissions. Only the lower 16 bits of `perm` are used.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_cmpnam()

Compare Two Names

Syntax

```
#include <strings.h>
#include <types.h>
error_code _os_cmpnam(
     char         *pattern,
     char         *string,
     u_int32      length,
     int32        *cmp_result);
```

Description

`_os_cmpnam()` compares a target name to a source pattern to determine if they are equal. `_os_cmpnam()` is not case-sensitive; it does not differentiate between upper and lower case characters.

`pattern` is a pointer to the pattern string. Two wildcard characters are recognized in the pattern string:

- A question mark (`?`) matches any single character.
- An asterisk (*) matches any string.

`string` is a pointer to the target name string. The target name must be terminated by a null byte.

`length` specifies the length of the pattern string.

`cmp_result` is a pointer to the location where `_os_cmpnam()`stores the lexicographic result of the comparison. `_os_cmpnam()` returns zero if the target string is the same as the pattern string.

OS-9 for 68K: If the error `EOS_PNNF` is returned, the strings differ and the integer at the address passed in `cmp_result` is not changed.

OS-9: If the error `EOS_DIFFER` is returned, the strings differ and the integer at `cmp_result` has the following meaning:

Table 2-17 _os_cmpnam() Parameter cmp_result Values

Integer	Description
`positive`	The target name is less than the pattern string.
`negative`	The target string is greater than the pattern string.

Note

On OS-9 for 68K, you never receive an `EOS_DIFFER` error. On OS-9, you never receive an `EOS_PNNF` error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$CmpNam`	***OS-9 for 68K Technical Manual***
`F_CMPNAM`	***OS-9000 Technical Manual***

_os_config()

Configure an Element

Syntax

```
#include <sysglob.h>
error_code _os_config(
     u_int32      code,
     void         *param);
```

Description

`os_config()` is a wildcard call that configures elements of the operating system which may or may not be process specific. It dynamically reconfigures operating system resources at runtime. The target resources may be system-wide resources or they may be process-specific, depending on the nature of the configuration call being made.

`code` is the configuration command code. Reference the ***OS-9 Technical Manual*** for a list of valid codes for your version of the operating system.

`param` is a pointer to any additional parameters required by the specified configuration function.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

See Also

`_os_ioconfig()`

_os_cpy_ioproc()

Get Pointer to I/O Process Descriptor

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_cpy_ioproc(
     process_id   proc_id,
     void         *buffer,
     u_int32      count);
```

Description

`_os_cpy_ioproc()` copies the I/O process descriptor for the specified process into a buffer.

`process_id` is the process ID.

`buffer` is a pointer to the buffer in which to copy the process descriptor.

`count` specifies the number of bytes to copy.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_cpymem()

Copy External Memory

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_cpymem(
     process_id   proc_id,
     void         *from,
     void         *to,
     u_int32      count);
```

Description

`_os_cpymem()` copies external memory into your buffer for inspection.

`proc_id` specifies the process ID of the owner of the external memory.

`from` is a pointer to the external process' memory to copy.

`to` is a pointer to the caller's destination buffer.

`count` is the number of bytes to copy.

`_os_cpymem()` can copy portions of the system's address space. This is especially helpful in examining modules. Any memory in the system may be viewed in this way.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_move()`

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_crc(
     void         *start,
     u_int32      count,
     int32        *accum);
```

Description

`_os_crc()` generates or checks the CRC (cyclic redundancy check) values of sections of memory. Compilers, assemblers, and other module generators use `_os_crc()` to generate a valid module CRC.

`start` is a pointer to the section of memory.

`count` specifies the byte count for the section of memory.

`accum` is a returned value. It is a pointer to the CRC accumulator.

If the CRC of a new module is to be generated, the CRC is accumulated over the module (excluding the CRC). The accumulated CRC is complemented and stored in the correct position in the module.

The CRC is calculated over a specified number of bytes starting at the source address. It is not necessary to cover an entire module in one call, because the CRC may be accumulated over several calls. The CRC accumulator must be initialized to `-1` before the first `_os_crc` call for any particular module.

To generate the CRC of an existing module, the calculation must be performed on the entire module, including the module CRC. The CRC accumulator contains the CRC constant bytes if the module CRC is correct. The CRC constant is defined in `module.h` as `CRCCON`. The value is `0x00800fe3`.

To generate the CRC for a module:

1. Initialize the accumulator to `-1`.

2. Perform the CRC over the module.
3. Call F_CRC with a NULL value for `start`.
4. Complement the CRC accumulator.
5. Write the contents of the accumulator to the module.

The CRC value is three bytes long, in a four-byte field. To generate a valid module CRC, you must include the byte preceding the CRC in the check. You must initialize this byte to zero. For convenience, if a data pointer of zero is passed, the CRC is updated with one zero data byte. `_os_crc` always returns `0xff` in the most significant byte of the `accum` parameter, so `accum` may be directly stored (after complement) in the last 4 bytes of a module as the correct CRC.

For More Information

Refer to your operating system technical manual for more information on how your operating system uses CRCs for modules.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

F$CRC	***OS-9 for 68K Technical Manual***
F_CRC	***OS-9 Technical Manual***

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_create(
     char         *name,
     u_int32      mode,
     path_id      *path,
     u_int32      perm, ...);
```

Description

`_os_create()` creates a new file.

- On multifile devices, the new file is entered in the directory structure.
- On non-multifile devices, `create` is synonymous with `open`.

`name` is a pointer to the path name of the new file.

`mode` specifies the access mode. If data is to be written to the file, `mode` must have the `FAM_WRITE` or `S_IWRITE` bit set. Only the lower 16 bits of `mode` are used.

`path` is a pointer to the location where `_os_create()` stores the path number that identifies the file in subsequent I/O service requests until the file is closed.

`perm` specifies the attributes to use for the new file. Only the lower 16 bits of `perm` are used.

If either the `FAM_SIZE` bit or the `S_ISIZE` (initial file size) bit is set, you can also pass an initial file size estimate as a variable argument.

If either the `FAM_NOCREATE` bit or the `S_IEXCL` (exclusive) bit is set, an error is returned if `name` already exists. If neither `FAM_NOCREATE` nor `S_EXCL` are set, any existing file called `name` is deleted.

By default, if the pathlist specifies a file name that already exists, `_os_create()` recreates the file.

For More Information

Refer to your operating system technical manual for the available mode and attribute bits.

Note

On OS-9 for 68K, the binding for this call exists for compatibility. If speed is an issue, use `_os9_create()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_attach()`
`_os_open()`
`_os_close()`
`_os_makdir()`
`I$Create` ***OS-9 Technical Manual***
`I_CREATE` ***OS-9 Technical Manual***

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os9_create(
     char         *name,
     u_int32      mode,
     path_id      *path,
     u_int32      perm, ...);
```

Description

`_os9_create()` creates a new file.

- On multifile devices, the new file name is entered in the directory structure.
- On non-multifile devices, `create` is synonymous with `open`.

`name` is a pointer to the path name.

`mode` specifies the access mode. If data is to be written to the file, `mode` must have the write bit set. Only the lower 16 bits of `mode` are used.

`path` is a pointer to the location where `_os9_create()` stores the path number.

`perm` specifies the attributes to use for the new file. Only the lower 16 bits of `perm` are used.

If either the `FAM_SIZE` bit or the `S_ISIZE` (initial file size) bit is set, you can also pass an initial file size estimate as a variable argument.

For More Information

Refer to the ***OS-9 for 68K Technical Manual*** for the available mode and attribute bits.

By default, if the pathlist specifies an existing file name, `_os9_create()` returns an error.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_attach()`
`_os_close()`
`_os_makdir()`
`_os_open()`
`I$Create` ***OS-9 Technical Manual***

_os_crlk()

Create New Resource Lock Descriptor

Syntax

```
#include <lock.h>
#include <types.h>
error_code _os_crlk(lk_desc **lock);
```

Description

`_os_crlk()` creates a new resource lock descriptor. A resource lock descriptor is allocated and initialized to a free state (that is, not currently owned).

`lock` is a pointer to the location where `_os_crlk()` stores the pointer to the lock descriptor. `lock` is used as a handle to perform further operations on the lock.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`lock.l`

See Also

`F_ACQLK`	***OS-9 Technical Manual***
`F_CAQLK`	***OS-9 Technical Manual***
`F_CRLK`	***OS-9 Technical Manual***
`F_DDLK`	***OS-9 Technical Manual***
`F_DELLK`	***OS-9 Technical Manual***
`F_RELLK`	***OS-9 Technical Manual***
`F_WAITLK`	***OS-9 Technical Manual***

_os_datmod()

Create Data Module

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_datmod(
     char         *mod_name,
     u_int32      size,
     u_int16      *attr_rev,
     u_int16      *type_lang,
     u_int32      perm,
     void         **mod_exec,
     mh_data      **mod_head);
```

Description

`_os_datmod()` creates a data module with the specified attribute/revision and clears the data portion of the module. The module is created and entered into the system module directory.

`mod_name` is a pointer to the module name string.

`size` is the size of the data portion.

The module's desired attribute and revision are passed at the location pointed to by `attr_rev`. `_os_datmod()` also stores the module's attribute and revision at the location pointed to be `attr_rev`.

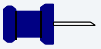
The module's desired type and language are passed at the location pointed to by `type_lang`. `_os_datmod()` also stores the module's actual type and language at the location pointed to by `type_lang`.

`perm` specifies the access permissions for the module. Only the lower 16 bits of `perm` are used.

`mod_exec` is a pointer to the location where `_os_datmod()` stores the pointer to the module data.

`mod_head` is a pointer to the location where `_os_datmod()` stores the pointer to the module header.

Be careful not to alter the data module's header or name string to prevent the module from becoming unknown to the system.



Note

The created module contains at least `size` usable data bytes but may be somewhat larger. The module itself is larger by at least the size of the module header, CRC, and name string; and it is rounded up to the nearest system memory allocation boundary.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_setcrc()`
`_os_move()`
`F$DatMod` ***OS-9 Technical Manual***
`F_DATMOD` ***OS-9 Technical Manual***

_os_ddlk()

Check for Deadlock Situation

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_ddlk(process_id proc_id);
```

Description

`_os_ddlk()` checks for a deadlock situation between processes. A search for the current process (calling process) in the linked list of potential conflictors is begun from the process specified by `proc_id`.

`proc_id` specifies the process with which to begin the search.

If the calling process is already in the linked list of processes, an `EOS_DEADLK` error is returned to the caller.

If the process is not in the linked list, the current process is added to the list associated with `proc_id`.

`_os_ddlk()` is useful for preventing a deadlock situation when protecting multiple resources from simultaneous accesses. The deadlock list usually represents the list of processes waiting to acquire access to an associated resource.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`lock.l`

See Also

F_ACQLK	***OS-9 Technical Manual***
F_CAQLK	***OS-9 Technical Manual***
F_CRLK	***OS-9 Technical Manual***
F_DDLK	***OS-9 Technical Manual***
F_DELLK	***OS-9 Technical Manual***
F_RELLK	***OS-9 Technical Manual***
F_WAITLK	***OS-9 Technical Manual***

_os9_delbit()

Deallocate in Bit Map

Syntax

```
#include <memory.h>
#include <types.h>
error_code _os9_delbit(
     u_int32      bit_number,
     u_int32      count,
     void         *address);
```

Description

`_os9_delbit()` clears bits in a buffer that represents a bit map. Bit numbers range from `0` to `n - 1`, where `n` is the number of bits in the bitmap.

`bit_number` specifies the bit number of the first bit to clear. Only the lower 16 bits of `bit_number` are used.

`count` specifies the number of bits to clear. Only the lower 16 bits of `count` are used.

`address` is a pointer to the base address of a bitmap.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os9_allbit()`
`_os_cpymem()`
`_os9_schbit()`
`F$DelBit`

OS-9 Technical Manual

_os_delete()

Delete File

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_delete(
     char         *name,
     u_int32      mode);
```

Description

`_os_delete()` deletes a file. The caller must have non-sharable write access to the file (the file may not already be open) or an error results. `name` is a pointer to the file to delete. `mode` specifies the access mode. Possible values for `mode` include:

```
FAM_READ FAM_WRITE FAM_EXEC
S_IREAD S_IWRITE S_IEXEC
```

Attempts to delete from non-multifile devices result in an error. The access mode specifies the data or execution directory (but not both) in the absence of a full pathlist. Only the lower 16 bits of `mode` are used.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

_os_attach()
_os_create()
_os_detach()
_os_open()
I$Delete *OS-9 Technical Manual*
I_DELETE *OS-9 Technical Manual*

_os_dellk()

Delete Existing Lock Descriptor

Syntax

```
#include <lock.h>
#include <types.h>
error_code _os_dellk(lk_desc *lock);
```

Description

`_os_dellk()` deletes an existing lock descriptor.

`lock` is a pointer to the `lk_desc` structure for the lock to delete.

`_os_dellk()` does not check for suspended processes still waiting to acquire the lock; an implementation using locks must release all processes waiting on a resource lock before deleting it. Failure to release suspended processes prior to deletion could result in system corruption.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`lock.l`

See Also

`F_ACQLK`	***OS-9 Technical Manual***
`F_CAQLK`	***OS-9 Technical Manual***
`F_CRLK`	***OS-9 Technical Manual***
`F_DDLK`	***OS-9 Technical Manual***
`F_DELLK`	***OS-9 Technical Manual***
`F_RELLK`	***OS-9 Technical Manual***
`F_WAITLK`	***OS-9 Technical Manual***

_os_delmdir()

Delete Existing Module Directory

Syntax

```
#include <moddir.h>
#include <types.h>
error_code _os_delmdir(char *name);
```

Description

`_os_delmdir()` deletes an existing module directory. The directory must be empty.

`name` is a pointer to the module directory.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F_DELMDIR`	***OS-9 Technical Manual***

_os_deltsk()

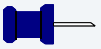
Deallocate Process Descriptor

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_deltsk(pr_desc *proc_desc);
```

Description

`_os_deltsk()` returns the process' protection resources when the process terminates. `proc_desc` is a pointer to the process descriptor.



Note

`_os_deltsk()` only performs a useful function on MMU/SSM systems. On non-MMU/SSM systems, `_os_deltsk()` simply returns without deallocating the process descriptor.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_alltsk()`
`_os_chkmem()`
`_os_protect()`
`_os_permit()`
`F$DelTsk` ***OS-9 for 68K Technical Manual***
`F_DELTSK` ***OS-9 Technical Manual***

_os_detach()

Remove Device from System

Syntax

```
#include <io.h>
#include <types.h>
error_code _os_detach(dev_list *dev_tbl);
```

Description

`_os_detach()` removes a device from the system device table if no other process is using the device.

`dev_tbl` is a pointer to the address of the device table entry.

Note

On OS-9 for 68K, if an invalid address is passed in `dev_tbl`, the system may crash or undergo severe damage.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_attach()`
`_os_close()`
`I$Detach` — ***OS-9 for 68K Technical Manual***
`I_DETACH` — ***OS-9 Technical Manual***

Syntax

```
#include <dexec.h>
#include <types.h>
error_code _os_dexec(
     process_id   proc_id,
     dexec_mode   mode,
     void         *params,
     u_int32      *num_instr,
     u_int32      num_bpts,
     void         **brk_pts,
     u_int32      *tot_instr,
     dexec_status *status,
     u_int32      *except,
     void         **addr,
     error_code   *exit_status);
```

Description

OS-9 for 68K:

`_os_dexec()` controls the execution of a suspended child process created by `_os_dfork()`. The process performing `_os_dexec()` is suspended, and its debugged child process is executed instead. This process terminates and control returns to the parent after the specified number of instructions have been executed, a breakpoint is reached, or an unexpected exception occurs. The parent and the child processes are never active at the same time.

OS-9:

`_os_dexec()` controls the execution of a suspended child process created by `F_DFORK`. The process performing the `F_DEXEC` is suspended, and its debugged child process is executed instead. This process terminates and control returns to the parent after the specified number of instructions have been executed, a breakpoint is reached, or an unexpected exception occurs. The parent and the child processes are never active at the same time.

`proc_id` is the process ID of the child to execute.

`mode` specifies the debug mode.

`params` is the parameter list pointer.

`num_instr` points to the location where the number of instructions to execute is passed. `_os_dexec()` also stores the number of instructions not executed at the location pointed to by `num_instr`. If the integer at `num_instr` is zero, commands are executed continuously.

`num_bpts` specifies the number of breakpoints in the list. Maximum of 16 break points allowed.

`brk_pts` is a pointer to the pointer to the breakpoint list.

`tot_instr` points to where the number of instructions executed so far is passed. `_os_dexec()` also stores the updated total number of instructions at the location pointed to by `tot_instr`.

`status` is a pointer to the location where `_os_dexec()` stores the return status.

`except` is a pointer to the location where `_os_dexec()` stores the exception that the child received.

`addr` is a pointer to the location where `_os_dexec()` stores the violation address.

`exit_status` is a pointer to the location where `_os_dexec()` stores the exit status of the child.

An `_os_dexit()` call must be made for the resources (memory) of the debugged process to be returned.

For More Information

Refer to your operating system technical manual for system-specific information.

Note

Tracing is allowed through subroutine libraries, intercept routines, and OS-9 for 68K user-state trap handlers. This is not a problem, but may seem strange at times.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_dfork()`	
`_os9_dfork()`	
`_os_dexit()`	
`F$DExec`	***OS-9 for 68K Technical Manual***
`F_DEXEC`	***OS-9 Technical Manual***
`F_DFORK`	***OS-9 Technical Manual***

_os_dexit()

Exit Debugged Program

Syntax

```
#include <dexec.h>
#include <types.h>
error_code _os_dexit(process_id proc_id);
```

Description

`_os_dexit()` terminates a suspended child process created by `_os_dfork()`. To allow examination of the child after its termination, normal termination by the child process does not release any resources.

`proc_id` is the process ID of the child to terminate.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_exit()`	
`_os_dfork()`	
`_os9_dfork()`	
`_os_dexec()`	
`F$DExit`	***OS-9 Technical Manual***
`F_DEXIT`	***OS-9 Technical Manual***

Syntax

```
#include <dexec.h>
#include <types.h>

error_code _os_dfork(
     u_int32      priority,
     u_int32      path_cnt,
     char         *mod_name,
     void         *params,
     u_int32      param_size,
     u_int32      mem_size,
     process_id   *proc_id,
     u_int32      type_lang,
     regs         *reg_stack,
     fregs        *fregstack);
```

Description

`_os_dfork()` creates a new process which becomes a child of the caller. It sets up the process' memory, MPU registers, and standard I/O paths. In addition, `_os_dfork()` allows a debugger utility to create a process whose execution can be closely controlled. The created process is not placed in the active queue, but is left in a suspended state. This allows the debugger to control its execution through the `_os_dexec()` and `_os_dexit()` calls.

`priority` is the priority of the new process. Only the lower 16 bits of `priority` are used.

`path_cnt` is the number of I/O paths for the child to inherit. Only the lower 16 bits of `path_cnt` are used.

`mod_name` is a pointer to the module name.

`params` is a pointer to the parameter block.

`param_size` specifies the size of the parameter block.

`mem_size` specifies any additional stack space to allocate. `mem_size` is specified in bytes.

`proc_id` is a pointer to the location where `_os_dfork()` stores the child process ID.

`type_lang` specifies the desired type and language. Only the lower 16 bits of `type_lang` are used.

`reg_stack` is a pointer to the register buffer.

`freg_stack` is a pointer to the floating point register buffer.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_dexit()`
`_os_fork()`
`_os9_dfork()`
`_os_dexec()`
`F$DExit` ***OS-9 Technical Manual***
`F_DFORK` ***OS-9 Technical Manual***

Syntax

```
#include <dexec.h>
#include <types.h>
error_code _os9_dfork(
     u_int32     priority,
     u_int32     path_cnt,
     char        *mod_name,
     void        *params,
     u_int32     param_size,
     u_int32     mem_size,
     process_id  *proc_id,
     u_int32     type_lang,
     void        *reg_buf);
```

Description

`_os9_dfork()` creates a new process which becomes a child of the caller. It sets up the new process' memory, MPU registers, and standard I/O paths. In addition, `_os9_dfork()` allows a debugger utility to create a process whose execution can be closely controlled. The process is not placed in the active queue, but is left in a suspended state. This allows the debugger to control its execution through the `_os_dexec()` and `_os_dexit()` system calls.

`priority` is the priority of the new process. Only the lower 16 bits of `priority` are used.

`path_cnt` is the number of the I/O paths for a child to inherit. Only the lower 16 bits of `path_cnt` are used.

`mod_name` is the module name pointer.

`params` is a pointer to an additional parameter block.

`param_size` is the size of the parameter block.

`mem_size` specifies any additional stack space to allocate. `mem_size` is specified in bytes.

`proc_id` is a pointer to the location where `_os9_dfork()` stores the child process ID.

`type_lang` specifies the desired type and language. Only the lower 16 bits of `type_lang` are used.

`reg_buf` is a pointer to a copy of all child registers.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_dexec()`
`_os_dexit()`
`_os_fork()`
`F$DFork` ***OS-9 for 68K Technical Manual***

Syntax

```
#include <dexec.h>
#include <types.h>
error_code _os_dforkm(
     u_int32     priority,
     u_int32     path_cnt,
     mh_com      *mod_head,
     void        *params,
     u_int32     param_size,
     u_int32     mem_size,
     process_id  *proc_id,
     regs        *reg_stack,
     fregs       *fregstack);
```

Description

`_os_dforkm()` creates a new process which becomes a child of the caller. It sets up the process' memory, MPU registers, and standard I/O paths. In addition, `_os_dforkm()` allows a debugger utility to create a process whose execution can be closely controlled. The created process is not placed in the active queue, but is left in a suspended state. This allows the debugger to control its execution through the `_os_dexec()` and `_os_dexit()` calls. `_os_dforkm()` is similar to `_os_dfork()`. However, `_os_dforkm()` is passed a pointer to the module to fork rather than the module name.

`priority` is the priority of the new process. Only the lower 16 bits of `priority` are used.

`path_cnt` is the number of I/O paths for the child to inherit. Only the lower 16 bits of `path_cnt` are used.

`mod_head` is a pointer to the module header.

`params` is a pointer to the parameter block.

`param_size` specifies the size of the parameter block.

`mem_size` specifies any additional stack space to allocate.

`proc_id` is a pointer to the location where `_os_dforkm()` stores the child process ID.

`reg_stack` is a pointer to the register buffer.

`fregstack` is a pointer to the floating point register buffer.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_fork()`
`_os_dexec()`
`_os_dexit()`
`_os_dfork()`

_os_dup()

Duplicate Path

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_dup(
     path_id     dup_path,
     path_id     *new_path);
```

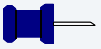
Description

`_os_dup()` returns a synonymous path number for an existing file or device.

`dup_path` is the path number of the path to duplicate.

`new_path` is a pointer to the location where `_os_dup()` stores the new path number for the same path.

`_os_dup()` always uses the lowest available path number. For example, if you perform an `_os_close()` on path `#0` and an `_os_dup()` on path `#4`, path `#0` is returned as the new path number. Therefore, service requests using either the old or new path numbers operate on the same file or device.



Note

It is usually not a good idea for more than one process to be performing I/O on the same path concurrently.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

I$Dup	***OS-9 for 68K Technical Manual***
I_DUP	***OS-9 Technical Manual***

_os_ev_allclr()

Wait for All Bits Defined by Mask to Become Clear

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_allclr(
     event_id     ev_id,
     int32        *value,
     signal_code  *signal,
     u_int32      mask);
```

Description

`_os_ev_allclr()` waits for an event to occur. That is, it waits until an `_os_ev_signal()` occurs that clears all of the bits corresponding to the set bits in the mask.

`ev_id` identifies the event.

`value` is a pointer to the location where `_os_ev_allclr()` stores the actual event value.

`signal` is a pointer to the location where `_os_ev_allclr()` stores the signal code. If the process receives a signal while in the event queue, it is activated even though the event has not actually occurred.

`mask` specifies the activation mask.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

F_EVENT	***OS-9 Technical Manual***
F_EVENT, EV_ALLCR	***OS-9 Technical Manual***
F_EVENT, EV_SIGNL	***OS-9 Technical Manual***

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_allset(
     event_id    ev_id,
     int32       *value,
     signal_code *signal,
     u_int32     mask);
```

Description

`_os_ev_allset()` waits for an event to occur. That is, it waits until an `_os_ev_signal()` occurs that sets all of the bits corresponding to the set bits in the mask. `ev_id` identifies the event. `value` is a pointer to the location where `_os_ev_allset()` stores the actual event value. `signal` is a pointer to the location where `_os_ev_allset()` stores the signal code. If the process receives a signal while in the event queue, it is activated even though the event has not actually occurred. `mask` specifies the activation mask.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F_EVENT`	***OS-9 Technical Manual***
`F_EVENT, EV_ALLSET`	***OS-9 Technical Manual***
`F_EVENT, EV_SIGNL`	***OS-9 Technical Manual***

_os_ev_anyclr()

Wait for Event to Occur

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_anyclr(
     event_id     ev_id,
     int32        *value,
     signal_code  *signal,
     u_int32      mask);
```

Description

`_os_ev_anyclr()` waits for an event to occur. It waits until an `_os_ev_signal()` occurs that clears any of the bits corresponding to the set bits in the mask. `ev_id` identifies the event. `value` is a pointer to the location where `_os_ev_anyclr()` stores the actual event value. `signal` is a pointer to the location where `_os_ev_anyclr()` stores the signal code. If the process receives a signal while in the event queue, it is activated even though the event has not actually occurred. `mask` specifies the activation mask.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F_EVENT`	***OS-9 Technical Manual***
`F_EVENT, EV_ANYCLR`	***OS-9 Technical Manual***
`F_EVENT, EV_SIGNL`	***OS-9 Technical Manual***

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_anyset(
     event_id     ev_id,
     int32        *value,
     signal_code  *signal,
     u_int32      mask);
```

Description

`_os_ev_anyset()` waits for an event to occur. That is, it waits until an `_os_ev_signal()` occurs that sets any of the bits corresponding to the set bits in the mask. `ev_id` identifies the event. `value` is a pointer to the location where `_os_ev_anyset()` stores the actual event value. `signal` is a pointer to the location where `_os_ev_anyset()` stores the signal code. If the process receives a signal while in the event queue, it is activated even though the event has not actually occurred. `mask` specifies the activation mask.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F_EVENT`	***OS-9 Technical Manual***
`F_EVENT, EV_ANYSET`	***OS-9 Technical Manual***
`F_EVENT, EV_SIGNL`	***OS-9 Technical Manual***

_os_ev_change()

Wait for Event to Occur

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_change(
     event_id     ev_id,
     int32        *value,
     signal_code  *signal,
     u_int32      mask,
     u_int32      pattern);
```

Description

`_os_ev_change()` waits for an event to occur. That is, it waits until an `_os_ev_signal()` occurs that changes any of the bits corresponding to the set bits in `mask`. `ev_id` identifies the event. `value` is a pointer to the location where `_os_ev_change()` stores the actual event value. `signal` is a pointer to the location where `_os_ev_change()` stores the signal code. If the process receives a signal while in the event queue, it is activated even though the event has not actually occurred. `mask` specifies the activation mask. `pattern` specifies the wait pattern.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F_EVENT`	***OS-9 Technical Manual***
`F_EVENT, EV_CHANGE`	***OS-9 Technical Manual***
`F_EVENT, EV_SIGNL`	***OS-9 Technical Manual***

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_creat(
     int32        winc,
     int32        sinc,
     u_int32      perm,
     event_id     *ev_id,
     char         *name,
     int32        value,
     u_int32      color);
```

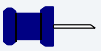
Description

`_os_ev_creat()` allows you to create events dynamically as needed. When an event is created, an initial value is specified, as well as increments to apply each time the event is waited for or an event signal occurs. Subsequent `_os_event` calls use the returned ID number to refer to the created event.

`winc` specifies the auto-increment value for `_os_ev_wait()`. Only the lower 16 bits of `winc` are used.

`sinc` specifies the auto-increment value for `_os_ev_signal()`. Only the lower 16 bits of `sinc` are used.

`perm` specifies the access permissions. Only the lower 16 bits of `perm` are used.



Note

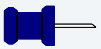
OS-9 for 68K ignores `perm`.

`ev_id` is a pointer to the location where `_os_ev_creat()` stores the event identifier.

`name` is a pointer to the event name string.

`value` specifies the initial event variable value.

`color` specifies the memory type for the event structure. Only the lower 16 bits of `color` are used.



Note

OS-9 for 68K ignores `color`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_ev_delete()`
`_os_ev_signal()`
`_os_ev_wait()`
`_os9_ev_wait()`

`F$Event`	***OS-9 for 68K Technical Manual*** or
`F_EVENT`	***OS-9 Technical Manual***
`F_EVENT, EV_CREAT`	***OS-9 Technical Manual***
`F_EVENT, EV_SIGNL`	***OS-9 Technical Manual***

_os_ev_delete()

Delete Existing Event

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_delete(char *name);
```

Description

`_os_ev_delete()` removes an event from the system event table, freeing the entry for use by another event. An event may not be deleted unless its use count is zero.

`name` is a pointer to the event's name string.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_ev_creat()`	
`F$Event`	***OS-9 for 68K Technical Manual***
`F_EVENT`	***OS-9 Technical Manual***
`F_EVENT, EV_DELET`	***OS-9 Technical Manual***
`F_EVENT, EV_SIGNL`	***OS-9 Technical Manual***

_os_ev_info()

Return Event Information

Syntax

```
#include <events.h>
#include <types.h>
```

OS-9 for 68K:

```
error_code _os_ev_info(
     event_id     ev_id,
     u_int32      size,
     ev_infostr   *buffer);
```

OS-9:

```
error_code _os_ev_info(
     event_id     ev_id,
     u_int32      size,
     void         *buffer);
```

Description

`_os_ev_info()` returns event information in the caller's buffer. The returned information is defined by the `ev_infostr` event information structure defined in the `events.h` header file.

`ev_id` specifies the event index to use to begin the search.

`size` specifies the buffer size.

`buffer` is a pointer to the buffer containing the event information.

On OS-9, the name of the event is appended to the end of the information structure; `buffer` and `size` must be large enough to accommodate the name of the target event.

`_os_ev_info()` returns the event information block for the first active event whose index is greater than or equal to this index. If no such event exists, an error is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Event`	***OS-9 for 68K Technical Manual OS-9 Technical Manual***
`F_EVENT`	***OS-9 for 68K Technical Manual***
`F_EVENT, EV_INFO`	***OS-9 for 68K Technical Manual***
`F_EVENT, EV_SIGNL`	***OS-9 for 68K Technical Manual***

_os9_ev_info()

Return Event Information

Syntax

```
include <events.h>
#include <types.h>
error_code _os9_ev_info(
     u_int32     *index,
     ev_infostr  *buffer);
```

Description

`os9_ev_info()` returns event information in the caller's buffer. The returned information is defined by the `ev_infostr` event information structure defined in `events.h`.

The event index to begin the search for a valid event is passed at the location pointed to by index. `_os9_ev_info()` also stores the event index of the valid event found (if any) at the location pointed to by index.

`buffer` is a pointer to a buffer large enough to hold the event information.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`os_lib.l`

See Also

`F$Event`	***OS-9 for 68K Technical Manual***

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_link(
     char         *name,
     event_id     *ev_id);
```

Description

`_os_ev_link()` determines the ID number of an existing event. To keep the use count synchronized properly, you must perform an `_os_ev_unlink()` call when the event is longer be used.

`name` is a pointer to the event name string.

`ev_id` is a pointer to the location where `_os_ev_link()` stores the event identifier.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_ev_unlink()`	
`F$Event`	***OS-9 for 68K Technical Manual***
`F_EVENT`	***OS-9 Technical Manual***
`F_EVENT, EV_LNK`	***OS-9 Technical Manual***
`F_EVENT, EV_UNLNK`	***OS-9 Technical Manual***

_os_ev_pulse()

Signal Event Occurrence

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_pulse(
     event_id     ev_id,
     int32        *value,
     u_int32      actv_flag);
```

Description

`_os_ev_pulse()` signals an event occurrence. The event value is set to what is passed at the location `value`, and the signal auto-increment is not applied. Then, the `_os_ev_signal()` search routine is executed and the original event value is restored.

`ev_id` identifies the event.

The value to pulse the event to is passed at the location pointed to by `value`. `_os_ev_pulse()` also stores the event value prior to the pulse at the location pointed to by `value`.

`actv_flag` specifies which process(es) to activate:

- If `actv_flag` is one, all processes in range are activated.
- If `actv_flag` is zero, the first process in the event queue waiting for that range is activated.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_ev_signal()`	
`F$Event`	***OS-9 for 68K Technical Manual***
`F_EVENT`	***OS-9 Technical Manual***
`F_EVENT, EV_PULSE`	***OS-9 Technical Manual***
`F_EVENT, EV_SIGNL`	***OS-9 Technical Manual***

_os_ev_read()

Read Event Value Without Waiting

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_read(
     event_id     ev_id,
     int32        *value);
```

Description

`_os_ev_read()` reads the value of an event without waiting for or affecting the event variable.

`ev_id` identifies the event.

`value` is a pointer to the location where `_os_ev_read()` stores the current event value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Event`	***OS-9 for 68K Technical Manual***
`F_EVENT`	***OS-9 Technical Manual***
`F_EVENT, EV_READ`	***OS-9 Technical Manual***
`F_EVENT, EV_SIGNL`	***OS-9 Technical Manual***

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_set(
     event_id     ev_id,
     int32        *value,
     u_int32      actv_flag);
```

Description

`_os_ev_set()` signals that an event has occurred. The current event variable is set to the value passed at `value`. Then, the event queue is searched for the first process waiting for that event.

`ev_id` identifies the event.

`value` is a pointer to the location where `_os_ev_set()` stores the event value prior to the set operation.

`actv_flag` specifies which process(es) to activate:

- If `actv_flag` is one, all processes in range are activated.
- If `actv_flag` is zero, the first process in the event queue waiting for that range is activated.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

_os_ev_signal()	
F$Event	***OS-9 for 68K Technical Manual***
F_EVENT	***OS-9 Technical Manual***
F_EVENT, EV_SET	***OS-9 Technical Manual***
F_EVENT, EV_SIGNL	***OS-9 Technical Manual***

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_setand(
     event_id     ev_id,
     int32        *value,
     u_int32      mask,
     u_int32      actv_flag)
```

Description

`_os_ev_setand()` signals that an event has occurred. The current event variable is ANDed with the value passed in `mask`. Then, the event queue is searched for the first process waiting for that event value.

`ev_id` identifies the event.

`value` is a pointer to the location where `_os_ev_setand()` stores the event value prior to the logical operation.

`mask` is the event mask.

`actv_flag` specifies which process(es) to activate:

- If `actv_flag` is one, all processes in range are activated.
- If `actv_flag` is zero, the first process in the event queue waiting for that range is activated.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

F_EVENT	***OS-9 Technical Manual***
F_EVENT, EV_SETAND	***OS-9 Technical Manual***
F_EVENT, EV_SIGNL	***OS-9 Technical Manual***

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_setor(
     event_id     ev_id,
     int32        *value,
     u_int32      mask,
     u_int32      actv_flag);
```

Description

`_os_ev_setor()` signals that an event has occurred. The current event variable is ORed with the value passed in `mask`. Then, the event queue is searched for the first process waiting for that event value.

`ev_id` identifies the event.

`value` is a pointer to the location where `_os_ev_setor()` stores the event value prior to the logical operation.

`mask` is the event mask.

`actv_flag` specifies which process(es) to activate:

- If `actv_flag` is one, all processes in range are activated.
- If `actv_flag` is zero, the first process in the event queue waiting for that range is activated.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

F_EVENT	***OS-9 Technical Manual***
F_EVENT, EV_SETOR	***OS-9 Technical Manual***
F_EVENT, EV_SIGNL	***OS-9 Technical Manual***

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_setr(
     event_id     ev_id,
     int32        *value,
     u_int32      actv_flag);
```

Description

`_os_ev_setr()` signals that an event has occurred. The current event value is incremented by `value`. Then, the event queue is searched for the first process waiting for that event value. Arithmetic underflows or overflows are set to `0x80000000` (minimum integer) or `0x7fffffff` (maximum integer), respectively.

`ev_id` identifies the event.

The amount to add to the event value is passed at the location pointed to by `value`.

- **OS-9 for 68K:** `_os_ev_setr()` stores the event value **before** the increment at `value`.
- **OS-9:** `_os_ev_setr()` stores the event value **after** the increment at `value`.

`actv_flag` specifies which process(es) to activate:

- If `actv_flag` is one, all processes in range are activated.
- If `actv_flag` is zero, the first process in the event queue waiting for that range is activated.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

_os_ev_set()
_os_ev_signal()
F$Event — ***OS-9 for 68K Technical Manual***
F_EVENT, EV_SETR — ***OS-9 Technical Manual***
F_EVENT, EV_SIGNL — ***OS-9 Technical Manual***

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_setxor(
     event_id     ev_id,
     int32        *value,
     u_int32      mask,
     u_int32      actv_flag);
```

Description

`_os_ev_setxor()` signals that an event has occurred. The current event value is **EXCLUSIVE-OR**ed with `mask`. Then, the event queue is searched for the first process waiting for that event value.

`ev_id` identifies the event.

`value` is a pointer to the location where `_os_ev_setxor()` stores the event value prior to the logical operation.

`mask` specifies the event mask.

`actv_flag` specifies which process(es) to activate:

- If `actv_flag` is one, all processes in range are activated.
- If `actv_flag` is zero, the first process in the event queue waiting for that range is activated.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

F_EVENT	***OS-9 Technical Manual***
F_EVENT, EV_SETXOR	***OS-9 Technical Manual***
F_EVENT, EV_SIGNL	***OS-9 Technical Manual***

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_signal(
     event_id     ev_id,
     int32        *value,
     u_int32      actv_flag);
```

Description

`_os_ev_signal()` signals that an event has occurred. The current event variable is updated with the signal auto-increment that was specified when the event was created. Then, the event queue is searched for the first process waiting for that event value.

`ev_id` identifies the event that has occurred.

`value` is a pointer to the location where `_os_ev_signal()` stores the event value prior to the signal operation.

`actv_flag` specifies which process(es) to activate:

If `actv_flag` is one, all processes in the event queue with a value in range are activated.

If `actv_flag` is zero, the first process in the event queue waiting for that range is activated.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Event`	***OS-9 for 68K Technical Manual***
`F_EVENT`	***OS-9 Technical Manual***
`F_EVENT, EV_SIGNL`	***OS-9 Technical Manual***

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_tstset(
     event_id    ev_id,
     int32       *value,
     signal_code *signal,
     u_int32     mask);
```

Description

`_os_ev_tstset()` waits for an event to occur. The event variable is ANDed with the value in `mask`. If the result is not zero, the calling process is suspended in a FIFO event queue until an `_os_ev_signal()` occurs that clears all of the bits corresponding to the set bits in the mask. Then, the bits corresponding to the set bits in the mask are set.

`ev_id` identifies the event.

`value` is a pointer to the location where `_os_ev_tstset()` stores the actual event value.

`signal` is a pointer to the location where `_os_ev_tstset()` stores the signal code. If a process in the event queue receives a signal, it is activated even though the event has not actually occurred.

`mask` specifies the activation mask.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

F_EVENT	***OS-9 Technical Manual***
F_EVENT, EV_TSTSET	***OS-9 Technical Manual***
F_EVENT, EV_SIGNL	***OS-9 Technical Manual***

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_unlink(event_id ev_id);
```

Description

`_os_ev_unlink()` decrements the use count of an event. When the use count reaches zero, you can delete the event using `_os_ev_delete()`.

`ev_id` identifies the event.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_ev_delete()`
`_os_ev_link()`

`F$Event`	***OS-9 for 68K Technical Manual***
`F_EVENT`	***OS-9 Technical Manual***
`F_EVENT, EV_UNLNK`	***OS-9 Technical Manual***
`F_EVENT, EV_DELET`	***OS-9 Technical Manual***

_os_ev_wait()

Wait for Event to Occur

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_wait(
     event_id     ev_id,
     int32        *value,
     signal_code  *signal,
     int32        min_val,
     int32        max_val);
```

Description

`_os_ev_wait()` waits for an event to occur. It waits until an event call places the value in the range between the minimum and maximum activation values. Then, the wait auto-increment (specified at creation) is added to the event variable.

`ev_id` identifies the event.

`value` is a pointer to the location where `_os_ev_wait()` stores the actual event value.

`signal` is a pointer to the location where `_os_ev_wait()` stores the signal code. If a process in the event queue receives a signal, it is activated even though the event has not actually occurred.

`min_val` is the minimum activation value.

`max_val` is the maximum activation value.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

_os_ev_signal()	
_os_ev_waitr()	
_os9_ev_waitr()	
F_EVENT	***OS-9 Technical Manual***
F_EVENT, EV_WAIT	***OS-9 Technical Manual***
F_EVENT, EV_SIGNL	***OS-9 Technical Manual***

_os9_ev_wait()

Wait for Event to Occur

Syntax

```
#include <events.h>
#include <types.h>
error_code _os9_ev_wait(
     event_id     ev_id,
     int32        *value,
     int32        min_val,
     int32        max_val);
```

Description

`_os9_ev_wait()` waits for an event to occur. It waits until an event call places the value in range between the minimum activation value and the maximum activation value. Then, the wait auto-increment (specified at creation) is added to the event variable. `ev_id` specifies the event. `value` is a pointer to the location where `_os9_ev_wait()` stores the actual event value. `min_val` specifies the minimum activation value. `max_val` specifies the maximum activation value.

If a process in the event queue receives a signal, it is activated even though the event has not actually occurred.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Event` ***OS-9 for 68K Technical Manual***

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_waitr(
     event_id    ev_id,
     int32       *value,
     signal_code  *signal,
     int32       *min_val,
     int32       *max_val);
```

Description

`_os_ev_waitr()` waits for an event to occur. It waits until an event call places the value in the range between the minimum and maximum activation values, where the integers at locations `min_val` and `max_val` are relative to the current event value. Then, the wait auto-increment (specified at creation) is added to the event variable.

`ev_id` identifies the event.

`value` is a pointer to the location where `_os_ev_waitr()` stores the actual event value.

`signal` is a pointer to the location where `_os_ev_waitr()` stores the signal code if a signal activates the process.

The minimum relative activation value is passed at the location pointed to by `min_val`. `min_val` is also a pointer to location where `_os_ev_waitr()` stores the minimum absolute activation value.

The maximum relative activation value is passed at the location pointed to by `max_val`. `max_val` is also a pointer to the location where `_os_ev_waitr()` stores the maximum absolute activation value.

The event value is added to `min_val` and `max_val`, and the actual values are returned to the caller. If an underflow or overflow occurs on the addition, the values `0x80000000` (minimum integer) and `0x7fffffff` (maximum integer) are used, respectively.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_ev_signal()`
`_os_ev_wait()`
`_os9_ev_waitr()`

`F_EVENT`	***OS-9 Technical Manual***
`F_EVENT, EV_SETOR`	***OS-9 Technical Manual***
`F_EVENT, EV_SIGNL`	***OS-9 Technical Manual***

Syntax

```
#include <events.h>
#include <types.h>
error_code _os9_ev_waitr(
     event_id    ev_id,
     int32       *value,
     int32       *min_val,
     int32       *max_val);
```

Description

`_os9_ev_waitr()` waits for an event to occur. It waits until an event call places the value in the range between the minimum and maximum activation values, where `min_val` and `max_val` are relative to the current event value. Then, the wait auto-increment (specified at creation) is added to the event variable.

`ev_id` specifies the event.

`value` is a pointer to the location where `_os9_ev_waitr()` stores the actual event value.

The minimum relative activation value is passed at the location pointed to by `min_val`. `min_val` is also a pointer to the location where `_os9_ev_waitr()` stores the minimum absolute value.

The maximum relative activation value is passed at the location pointed to by `max_val`. `max_val` is also the pointer to the location where `_os9_ev_waitr()` stores the maximum absolute value.

The event value is added to `min_val` and `max_val` respectively, and the actual values are returned to the caller. If an underflow or overflow occurs on the addition, the values `$80000000` (minimum integer) and `$7fffffff` (maximum integer) are used, respectively.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os9_ev_wait()`
`F$Event`
OS-9 for 68K Technical Manual

Syntax

```
#include <process.h>
#include < dexec.h>  /* used when _os_dfork(),
                    /* _os9_dfork(), or _os_dforkm()
                     /* is specified as func */
#include <types.h>
error_code _os_exec(
     u_int32 (*func)(),
     u_int32 priority,
     u_int32 pathcnt,
     void *modname,
     char **argv,
     char **envp,
     u_int32  datasize, ...);
```

Description

`_os_exec()` prepares the parameter and environment list before creating a process. The function to create a new process passes information to the new process as binary data specified by a pointer and size. It is up to the forked process to interpret the data.

`func` is a pointer to one of the following functions that create a new process:

- **OS-9 for 68K:** `_os_fork()`, `_os9_dfork()`, or `_os_chain()`
- **OS-9:** `_os_fork()`, `_os_forkm()`, `_os_chain()`, `_os_chainm()`, `_os_dfork()`, or `_os_dforkm()`

`priority` is the priority value at which the new process is to run. Only the lower 16 bits of `priority` are used. If `priority` is zero, the new process receives the priority of the calling process.

`pathcnt` is the number of open paths to pass to the new process. Normally, `pathcnt` is three, causing the three standard paths to be passed. A value of zero passes **no** open paths. Only the lower 16 bits of `pathcnt` are used.

modname is a pointer to a string naming the new primary module or a pointer to the module in the case of the functions ending in m (_os_forkm(), _os_chainm(), and _os_dforkm()).

argv is a pointer to the parameter pointer list that the new process is to receive. By convention, argv[0] is the name of the new module. The end of the argv list is marked by a null pointer.

envp is a pointer to the environment pointer list containing environment variables that the new process is to receive. The end of the envp list is also marked by a null pointer.

datasize is the additional memory (in bytes) to allocate to the new process' stack memory. A datasize of zero indicates the default stack size required by the process.

The first seven parameters are identical for _os_fork(), _os_forkm(), _os_chain(), _os_chainm(), _os_dfork(), or _os_dforkm(). Four additional parameters may be required (add1, add2, add3, and add4) and have different meaning depending on the function specified by func:

Table 2-18 _os_exec() Additional Parameters

Call	Parameter	Definition
_os_fork()	add1	Child process ID returns to the location of the add1 pointer
	add2	Type/language of the new process
	add3	*Orphan: 0 = Normal child process 1 = Process not associated with parent

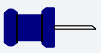
Table 2-18 _os_exec() Additional Parameters (continued)

Call	Parameter	Definition
_os_forkm()	add1	The child process ID returns to the location of the add1 pointer
	add2	*Orphan: 0 = Normal child process 1 = Process not associated with parent
_os_chain()	add1	Type/language of the new process
_os_chainm()		Does not use the extra parameters
_os_dfork()	add1	Child process ID returns to the location of the add1 pointer
	add2	Type/language of the new process
	add3	Pointer to register stack for child process
	add4	Pointer to floating point register stack for child process
_os9_dfork()	add1	Child process ID returns to the location of the add1 pointer
	add2	Type/language of the new process
	add3	Pointer to register stack for child process

Table 2-18 _os_exec() Additional Parameters (continued)

Call	Parameter	Definition
`_os_dforkm()`	`add1`	Child process ID returns to the location of the `add1` pointer
	`add2`	Pointer to register stack for child process
	`add3`	Pointer to floating point register stack for child process

*On OS-9 for 68K, the `orphan` flag must be set to zero.



Note

If you are using `_os_exec()` when `func` indicates a chain function, you must provide a full pathlist for the module or verify that the module is already in memory.

`_os_exec()` returns the value of `(*func)()`. It returns `0` if successful. Otherwise, it returns an error number.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

_os_chain()
_os_chainm()
_os_dfork()
_os_dforkm()
_os_fork()
_os_forkm()
os9fork(), os9forkc()
os9exec()
os9kexec()

Example

```
#include <process.h>
#include <dexec.h>
#include <errno.h>
#include <cglob.h>
#include <types.h>
                        /* Use the environment pointer given */
                        /* to you by the shell.            */
char *argblk[] =
{
   0,0,0,0,0,0,         /* Create an array of pointers to  */
};                      /* become argv[] of child process. */
main()
{
  u_int16 prior = 0;    /* Priority of Child process  */
                        /* 0 is same as parent        */
  u_int16 paths = 3;    /* number of paths to inherit */
                        /* stdin, stdout, stderr      */
  u_int32 edata = 0;    /* No Extra Data Space        */
  process_id child_id; /* child id number            */
  char  orphan = 0;     /* make this child a normal   */
                        /* child not an orphan        */
  u_int16 type_lang;    /* type language of child     */

  type_lang  =  mktypelang(MT_PROGRAM, ML_OBJECT);

  argblk[0] = "dir";    /* Child process is "dir" */
  argblk[1] = "-e";     /* With a -e option */
  argblk[2] = "-a",     /* And a -a option */
  argblk[3] = 0;     /* Make sure end of options is a null ptr */
```

```
  if ((errno = _os_exec(_os_fork, prior, paths,
                        argblk[0], argblk, _environ,
                        edata, &child_id, type_lang,
                        orphan)) != SUCCESS)
     _os_exit(_errmsg(errno,
              "Problem Creating Child %s\n",
               argblk[0]));

/*If we get this far, the _os_exec() was successful*/
 _os_ wait(0);      /* Wait for child process to die */
/* die*/
   printf("The Child's ID number was %d\n",
           child_id);
}
```

Syntax

```
int os9exec(
     int          (*func)(),
     char         *modname,
     char         **argv,
     char         **envp,
     unsigned     stacksize,
     short        priority,
     short        pathcnt);
```

Description

`os9exec()` is a high-level fork/chain interface that prepares the parameter and environment list before a process is created.

`os9forkc()` and `chainc()` are C-level hooks to the system calls. By themselves, they provide no real parameter information. This leaves the interpretation of a raw parameter string up to the `cstart` portion of the C program. Problems can occur when handling quotes and control characters.

`os9exec()` provides more precise control over the parameter strings and enables the environment variables to be easily inherited from the parent process. `os9exec()` closely resembles the UNIX `execve()` system call.

`func` is a pointer to a function that creates a new process. This function is normally `chain()`, `chainc()`, `os9fork()`, or `os9forkc()`. `func` passes information to a new process as binary data specified by a pointer and size. It is up to the forked process to interpret the data.

`modname` is a pointer to a string naming the new primary module.

`argv` is a pointer to the parameter pointer list that the new process receives. By convention, `argv[0]` is the name of the new module and a null pointer marks the end of the `argv` list.

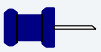
envp is a pointer to the environment pointer list that the new process receives. A null pointer marks the end of the envp list. You can set envp to point to the environment list of the current process to allow the child to inherit that environment.

stacksize specifies the additional memory (in bytes) to allocate to the new process' stack memory. A stacksize of zero indicates the default stack size the process requires.

priority is the priority at which the new process runs. If priority is zero, the new process receives the priority of the calling process.

pathcnt is the number of open paths to pass to the new process.

If func is os9forkc() or os9fork() and the initial attempt to create the child process fails due to EOS_PNNF (path name not found), os9exec() calls modloadp() with modname to load the module from disk and attempts the fork again. This action is the same as the shell handling of executable files.

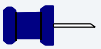


Note

If you use os9exec() when func is chain() or chainc(), you must either:

- Ensure that the module is in memory or in your current execution directory. (See modlink(), modload(), and modloadp().)
- Provide a full pathlist to the module.

os9exec() returns the value of (*func)(). This is the ID of the child process for os9fork(). -1 is returned on error.



Note

Any program that executes a C program must do so using the `os9exec()` interface. The `cstart` module can handle parameter strings passed by `os9fork()` and `chain()`, but no environment is available and the `argv` pointer list is separated by white space.

Note

`os9exec()` is not automatically available if you are using ANSI or extended-ANSI modes. To use `os9exec()` in one of these modes, link with `sys_clib.l`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

For More Information

The primary purpose of the `sys_clib.l` library is to provide a library for compatibility with the Microware K&R C compiler. For additional information, refer to Appendix A: Prototyping sys_clib.l Functions.

See Also

chainc(), chain()
modlink()
modload()
modloadp()
_os_exec()
os9fork(), os9forkc()

Example

The following code fragment is an example call to os9exec().

The argblk array contains pointers to the parameters in the order that the new program is to receive them. The new process receives a copy of the parameters, not the addresses of the parameters.

This example passes the current environment by naming the global variable _environ.

```
extern int os9forkc();
extern char **_environ;

char *argblk[] = {
     "rename",
     "-x",
     "oldname",
     "newname",
     0,
};

main ()
{
    int pid;
    if ((pid = os9exec(os9forkc,argblk[0],argblk,
        _environ,0,0,3)) > 0) wait(0);
    else printf ("can't fork %s",argblk[0]);
}
```

_os_exit()

Terminate Calling Process

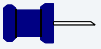
Syntax

```
#include <process.h>
#include <types.h>
error_code _os_exit(status_code status);
```

Description

`_os_exit()` allows a process to terminate itself. Its data memory area is deallocated and its primary module is unlinked. All open paths are automatically closed.

`status` is the status code returned to the parent process.



Note

The parent **must** perform an `_os_wait()` or an `_os_exit()` before the process descriptor is returned to free memory.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

_os_aproc()	
_os_close()	
_os_findpd()	
_os_fork()	
_os9_retpd()	
_os_srtmem()	
_os_unlink()	
_os_wait()	
F$Exit	*OS-9 for 68K Technical Manual*
F_EXIT	*OS-9 Technical Manual*

_os_findpd()

Find Process Descriptor

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_findpd(
     process_id   proc_id,
     pr_desc      **proc_desc);
```

Description

`_os_findpd()` converts a process number to the absolute address of its process descriptor data structure.

`proc_id` specifies the process ID.

`proc_desc` is a pointer to the location where `_os_findpd()` stores the pointer to the process descriptor.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_alocproc()`	
`_os_rtnprc()`	
`F$FindPD`	***OS-9 for 68K Technical Manual***
`F_FINDPD`	***OS-9 Technical Manual***

_os9_findpd()

Find Fixed Block of Memory

Syntax

```
#include <process.h>
#include <types.h>
error_code _os9_findpd(
     u_int32      number,
     void         **address,
     void         *table);
```

Description

`_os9_findpd()` converts a table index to the absolute address of its data structure.

`number` is the number of the table element. Only the lower 16 bits of `number` are used.

`address` is a pointer to the location where `_os9_findpd()` stores the table entry pointer.

`table` is a pointer to the table to use.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os9_allpd()`
`_os9_retpd()`
`F$FindPD` ***OS-9 for 68K Technical Manual***

Syntax

```
#include <regs.h>
error_code _os_firq(
     u_int32     vector,
     u_int32     priority,
     void        *irq_entry,
     void        *statics);
```

Description

`_os_firq()` installs an IRQ service routine into or removes one from the system's fast IRQ system.

`vector` specifies the vector number.

`priority` specifies the priority. `priority` is reserved and must be zero.

`irq_entry` is a pointer to the IRQ service routine's entry point. If `irq_entry` is zero, the IRQ service routine is deleted.

`statics` is a pointer to the global static storage.

This fast IRQ system provides a faster interrupt response context than the normal IRQ vector polling scheme (provided through `_os9_irq()`).

- To place a routine into the fast IRQ system, set `irq_entry` to a non-zero value.
- To remove a routine from the fast IRQ system, set `irq_entry` to zero.

Only one `_os_firq()` routine can be active at a time per vector. An attempt to install a second routine on a vector using `_os_firq()` causes an E0S_VCTBSY error. If additional devices are required to be on the same vector as an `_os_firq()` device, use `_os9_irq()` to install them.

The interrupt stack provided is a small “default” IRQ stack. The service routine must ensure that this stack is not overflowed. The `init` module’s “IRQ stack” value does **not** affect this stack; the `init` module value is used to control the `_os9_irq()` polling system stack.

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_irq()`
`F$FIRQ` ***OS-9 for 68K Technical Manual***

_os_fmod()

Find Module Directory Entry

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_fmod(
     u_int32      *type_lang,
     mod_dir      **moddir_entry,
     char         *mod_name);
```

Description

`_os_fmod()` searches the module directory for a module whose name, type, and language match the parameters. If found, a pointer to the module directory entry is returned in `moddir_entry`.

`type_lang` specifies the type and language of the module. Only the lower 16 bits of `type_lang` are used.

`moddir_entry` is a pointer to the location where `_os_fmod()` stores the pointer to the module directory entry.

`mod_name` is a pointer to the module name.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_fork()

Create New Process

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_fork(
     u_int32      priority,
     u_int32      path_cnt,
     char         *mod_name,
     void         *params,
     u_int32      param_size,
     u_int32      mem_size,
     process_id   *proc_id,
     u_int32      type_lang,
     u_int32      orphan);
```

Description

`_os_fork()` creates a new process which becomes a child of the caller. It sets up the new process' memory, MPU registers, and standard I/O paths.

`priority` specifies the priority of the new process. Only the lower 16 bits of `priority` are used. If `priority` is zero, the new process inherits the same priority as the calling process.

`path_cnt` specifies the number of I/O paths for the child to inherit. Only the lower 16 bits of `path_cnt` are used.

`mod_name` is the module name pointer.

`params` is a pointer to the parameter block.

`param_size` specifies the size of the parameter block.

`mem_size` specifies any additional stack space to allocate.

`proc_id` is a pointer to the location where `_os_fork()` stores the child process ID.

`type_lang` specifies the desired type and language. Only the lower 16 bits of `type_lang` are used. If `type_lang` is zero, any module may be loaded regardless of type and language.

If the `orphan` flag is non-zero, the new process executes without a parent. If the `orphan` flag is zero, the new process is the child of the calling process. On OS-9 for 68K if the `orphan` flag is set to a non-zero value, the `EOS_PARAM` error is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_wait()`
`_os_exit()`
`_os_chain()`

`F$Fork`	***OS-9 Technical Manual***
`F_FORK`	***OS-9 Technical Manual***

Syntax

```
#include <process.h>
#include <types.h>

int os9fork(
     char         *modname,
     int          parmsize,
     char         *parmptr,
     short        type,
     short        lang,
     int          datasize,
     short        prior);

int os9forkc(
     char         *modname,
     int          parmsize,
     char         *parmptr,
     short        type,
     short        lang,
     int          datasize,
     short        prior,
     short        pathcnt);
```

Description

`os9forkc()` creates a process that runs concurrently with the calling process. When the new process terminates, its exit status is available to the forking (parent) process. The new process inherits the standard paths (0, 1, and 2) plus any additional paths indicated by `pathcnt`.

`modname` is a pointer to a null-terminated module name.

`parmsize` specifies the size of the parameter list. Generally, you must set `parmsize` equal to `strlen(parmptr)+1`.

`parmptr` is a pointer to a null-terminated string to pass to the new process.

`type` specifies the type of module. If `type` is zero, any module may be forked, regardless of type.

`lang` specifies the module language. If `lang` is zero, any module may be forked, regardless of language.

`datasize` gives extra memory to the new program. If no extra memory is required, set `datasize` to zero.

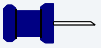
`prior` is the new priority at which to run the program. Indicate zero if no priority change is desired.

`pathcnt` is the number of the parent's open paths for the new process to inherit.

If the fork is unsuccessful, `-1` is returned and the appropriate error code is placed in the global variable `errno`. If the fork is successful, the process ID number of the new process is returned.

`os9fork()` is the same as `os9forkc()` without a `pathcnt` parameter.

`_os_exec()` is the preferred method to fork C programs.



Note

Beware of forking to a system module. It only makes sense to fork to a program of type object module. `os9fork()` is a historical function and is likely to be removed in a future release. Use `os9forkc()` instead.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

For More Information

The primary purpose of the `sys_clib.l` library is to provide a library for compatibility with the Microware K&R C compiler. For additional information, refer to Appendix A: Prototyping sys_clib.l Functions.

See Also

`_os_chain()`	
`_os_exec()`	
`_os_fork()`	
`F$Chain`	***OS-9 for 68K Technical Manual***
`F$Fork`	***OS-9 for 68K Technical Manual***
`F_CHAIN`	***OS-9 Technical Manual***
`F_FORK`	***OS-9 Technical Manual***

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_forkm(
     u_int32     priority,
     u_int32     path_cnt,
     mh_com      *mod_head,
     void        *params,
     u_int32     param_size,
     u_int32     mem_size,
     process_id  *proc_id,
     u_int32     orphan);
```

Description

`_os_forkm()` creates a new process which becomes a child of the caller. It sets up the new process' memory, MPU registers, and standard I/O paths. The new process is forked by a module pointer. `_os_forkm()` assumes that the module pointer is the primary module pointer for the new process.

`priority` specifies the priority of the new process. Only the lower 16 bits of `priority` are used. If `priority` is zero, the new process inherits the same priority as the calling process.

`path_cnt` specifies the number of I/O paths for the child to inherit. Only the lower 16 bits of `path_cnt` are used.

`mod_head` is the module name pointer.

`params` is a pointer to the parameter block.

`param_size` specifies the size of the parameter block.

`mem_size` specifies any additional stack space to allocate.

`proc_id` is a pointer to the location where `_os_forkm()` stores the child process ID.

If the `orphan` flag is non-zero, the new process executes without a parent. If the `orphan` flag is zero, the new process is the child of the calling process. Only the lower 16 bits of `orphan` are used.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_fork()`

_os_get_blkmap()

Get Free Memory Block Map

Syntax

```
#include <memory.h>
#include <types.h>
error_code _os_get_blkmap(
     void         *start,
     void         *buffer,
     u_int32      size,
     u_int32      *min_alloc,
     u_int32      *num_segs,
     u_int32      *tot_mem,
     u_int32      *free_mem);
```

Description

`_os_get_blkmap()` copies the address and size of the system's free RAM blocks into your buffer for inspection. It also returns information concerning the free RAM as noted by the parameters. A series of structures showing the address and size of free RAM blocks is returned in your buffer.

`start` is a pointer to the address to begin reporting the segments.

`buffer` is a pointer to the buffer to use.

`size` specifies the buffer size in bytes.

`min_alloc` is a pointer to the location where `_os_get_blkmap()` stores the minimum memory allocation size for the system.

`num_segs` is a pointer to the location where `_os_get_blkmap()` stores the number of memory fragments in the system.

`tot_mem` is a pointer to the location where `_os_get_blkmap()` stores the total RAM found by the system at startup.

`free_mem` is a pointer to the location where `_os_get_blkmap()` stores the current total free RAM available.

Note

Although `_os_get_blkmap()` returns the address and size of the system's free memory blocks, you cannot assume that these blocks are free to use. Use `_os_srqmem()` to request free memory blocks.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_mem()`
`_os_srqmem()`
`F$GBlkMap()` — ***OS-9 for 68K Technical Manual***
`F_GBLKMP` — ***OS-9 Technical Manual***

_os_getdl()

Get System I/O Device List Head Pointer

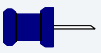
Syntax

```
#include <io.h>
#include <types.h>
error_code _os_getdl(dev_list **dev_list_ptr);
```

Description

`_os_getdl()` returns a pointer to the first entry in the system's I/O device list.

`dev_list_ptr` is a pointer to the location where `_os_getdl()` stores the pointer to the first entry in the device list.



Note

This pointer should never be accessed directly in user state. You should use `_os_cpymem()` to get a copy of the device list entry.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_getime()

Get System Date/Time

Syntax

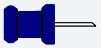
```
#include <time.h>
#include <types.h>
error_code _os_getime(
     time_t       *time,
     u_int32      *ticks);
```

Description

`_os_getime()` returns the current system time in the number of seconds since a date specified in Greenwich Mean Time. To determine the specified date, refer to the applicable manual, ***OS-9 for 68K Technical Manual*** or ***OS-9 Technical Manual***.

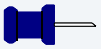
`time` is a pointer to the location where `_os_getime()` stores the current time.

`ticks` is a pointer to the location where `_os_getime()` stores the clock tick rate in ticks per second in the most significant word and the current tick in the least significant word.



Note

On OS-9 for 68K, the binding for this call performs conversions which allow compatibility with OS-9. If speed is a consideration, see `_os9_getime()`.



Note

`_os_getime()` returns a date and time of zero (with no error) if no previous call to `_os_setime()` has been made. A tick rate of zero indicates the clock is not running.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_setime()`
`_os9_getime()`

`F$Time`	***OS-9 for 68K Technical Manual***
`F_TIME`	***OS-9 Technical Manual***

_os9_getime()

Get System Date/Time

Syntax

```
#include <time.h>
#include <types.h>
error_code _os9_getime(
     u_int32     format,
     u_int32     *time,
     u_int32     *date,
     u_int16     *day,
     u_int32     *ticks);
```

Description

`_os9_getime()` returns the current system date and time. In the normal (Gregorian) format, `time` is expressed as `00hhmmss` and `date` as `yyyymmdd`. The Julian format expresses `time` as seconds since midnight and `date` as the Julian day number.

`format` specifies the format of the returned date and time:

Table 2-19 _os9_getime() Format Values

Format	The Date and Time Are Returned In
`0`	Gregorian format
`1`	Julian format
`2`	Gregorian format with ticks
`3`	Julian format with ticks

`time` is a pointer to the location where `_os9_getime()` stores the current system time.

`date` is a pointer to the location where `_os9_getime()` stores the current system date.

`day` is a pointer to the location where `_os9_getime()` stores the day of the week, where `0` is Sunday and `6` is Saturday.

`ticks` is a pointer to the location where `_os9_getime()` stores the clock tick data. If `ticks` are requested, the clock tick rate in ticks per second is returned in the most significant word at the location pointed to by `ticks`. The least significant word contains the current tick.

The following figures illustrate values returned at the pointers.

Figure 2-7 Gregorian Formats

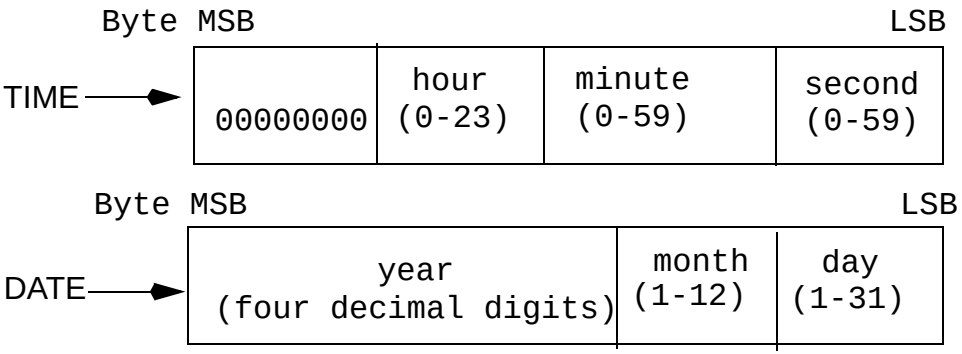
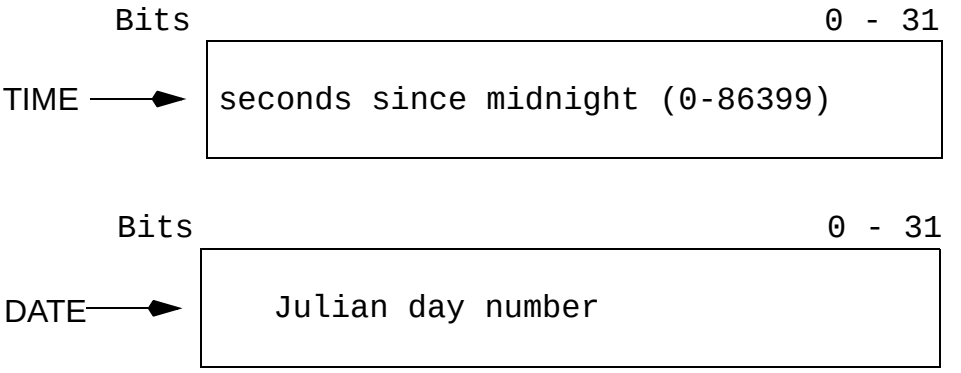


Figure 2-8 Julian Formats



Note

`_os9_getime()` returns a date and time of zero (with no error) if no previous call to `_os9_setime()` is made after the system start-up. A tick rate of zero indicates the clock is not running.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os9_setime()`
`_os_julian()`
`_os_getime()`

`F$Time` ***OS-9 for 68K Technical Manual***

_os_get_ioproc()

Get Pointer to I/O Process Descriptor

Syntax

```
#include <io.h>
#include <types.h>
error_code _os_get_ioproc(
     process_id   proc_id,
     io_proc      **proc);
```

Description

`_os_get_ioproc()` returns a pointer to the I/O process descriptor for the specified process.

`proc_id` specifies the process ID of the process.

`proc` is a pointer to the location where `_os_get_ioproc()` stores the pointer to the I/O process descriptor.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_get_mdp()

Get Current and Alternate Module Directory Pathlists

Syntax

```
#include <moddir.h>
#include <types.h>
error_code _os_get_mdp(
     char         *current,
     char         *alternate);
```

Description

`_os_get_mdp()` returns pathlists to the current and alternate module directories.

`current` is a pointer to the location where `_os_get_mdp()` stores the pathlist of the current module directory.

`alternate` is a pointer to the location where `_os_get_mdp()` stores the pathlist of the alternate module directory.

Note

`current` and `alternate` should point to buffers of at least 64 bytes to accommodate the directory names.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_get_moddir()

Get Copy of Module Directory

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_get_moddir(
     void         *buffer,
     u_int32      *count);
```

Description

`_os_get_moddir()` copies the process' current module directory into your buffer for inspection. `buffer` is a pointer to the buffer. The maximum number of bytes to copy is passed at the location pointed to by `count`. `_os_get_moddir()` also stores the actual number of bytes copied at the location pointed to by `count`.

Although the module directory contains pointers to each module in the system, never access the modules directly. Use `_os_cpymem()` to copy portions of the system's address space for inspection.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_cpymem()`
`_os_move()`

_os_getpd()

Find Path Descriptor

Syntax

```
#include <io.h>
#include <types.h>
error_code _os_getpd(
     path_id      path,
     pd_com       **pd);
```

Description

`_os_getpd()` converts the path number to the absolute address of its path descriptor data structure.

`path` specifies the path number.

`pd` is a pointer to the location where `_os_getpd()` stores the address of the path descriptor.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os9_allpd()`
`_os9_retpd()`

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_get_prtbl(
     void         *buffer,
     u_int32      *count);
```

Description

`_os_get_prtbl()` copies the process descriptor block table into the caller's buffer for inspection. `buffer` is a pointer to the buffer. The maximum number of bytes to copy is passed at the location pointed to by `count`. `_os_get_prtbl()` also stores the actual number of bytes copied at the location pointed to by `count`.

Although `_os_get_prtbl()` returns pointers to all process descriptors, never access the process descriptors directly. Instead, use `_os_gprdsc()` to inspect specific process descriptors.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_alocproc()`
`_os_gprdsc()`

_os_getstat()

Get File/Device Status

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_getstat(
     path_id      path,
     u_int32      code,
     void         *pb);
```

Description

`_os_getstat()` is a wildcard call used to handle individual device parameters that are not uniform on all devices or are highly hardware dependent.

`path` is the path number.

`code` is the `GetStat` code.

`pb` is a pointer to the parameter block.

The exact operation of this call depends on the device driver and file manager associated with the path.

The mnemonics for the status codes are located in the `sg_codes.h` header file.

The status codes that are currently defined by Microware and the functions that they perform are described in this manual. The functions have an `_os_gs_ prefix`.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_getsys()

Examine System Global Variable

Syntax

```
#include <sysglob.h>
#include <types.h>
error_code _os_getsys(
     u_int32     offset,
     u_int32     size,
     glob_buff   *sysvar);
```

Description

`_os_getsys()` enables a process to examine a system global variable. These variables have a `d_` prefix in the system header file `sysglob.h`. Consult the files in the DEFS directory for a description of the system global variables.

`offset` is the variable's offset in the system globals.

`size` specifies the size of the variable (1, 2, or 4 bytes).

`sysvar` is a pointer to the location where `_os_getsys()` stores the value of the requested system global.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Example

This example gets the current value of the tick counter:

```
#include <stddef.h>
if ((err = _os_getsys(offsetof(sysglobs, d_ticks),
     4, &buf)) != 0)
   exit(err);
```

See Also

_os_setsys()

_os_gprdsc()

Get Process Descriptor Copy

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_gprdsc(
     process_id   procid,
     void         *buffer,
     u_int32      *count);
```

Description

`_os_gprdsc()` copies a process descriptor into your buffer for inspection.

`procid` is the requested process ID.

`buffer` is a pointer to the buffer to use.

The maximum number of bytes to copy is passed at the location pointed to by `count`. `_os_gprdsc()` also stores the actual number of bytes copied at the location pointed to by `count`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_get_prtbl()`

_os_gregorian()

Get Gregorian Date

Syntax

```
#include <time.h>
#include <types.h>
error_code _os_gregorian(
     u_int32     *time,
     u_int32     *date);
```

Description

`_os_gregorian()` converts Julian dates to Gregorian dates. Gregorian dates are considered the normal calendar dates. `time` is a pointer to the location where the time in seconds since midnight. `_os_gregorian()` stores the time in the form `00hhmmss` in the location pointed to by `time`. `date` is a pointer to the location where the Julian date is passed. `_os_gregorian()` stores the date in the form `yyyymmdd` in the location pointed to by `date`. Refer to your operating system technical manual for more information.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_julian()`
`_os_getime()`

Syntax

```
#include <cdi.h>
#include <types.h>
error_code _os9_gs_cdfd(
     path_id     path,
     u_int32     count,
     void        *fdbuf);
```

Description

`_os9_gs_cdfd()` reads the file descriptor describing the path number. The file descriptor may be read for informational purposes only, as there are no user-changeable parameters.

`path` specifies the path number.

`count` is the number of bytes to copy. Only the lower 16 bits of `count` are used.

`fdbuf` is a pointer to the buffer area for the file descriptor.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_cpypd()

Copy Contents of Path Descriptor

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_cpypd(
     path_id      path,
     u_int32      *size,
     void         *path_desc);
```

Description

`_os_gs_cpypd()` copies the contents of the specified path's path descriptor to your buffer.

`path` is the path number.

The number of bytes to copy from the path descriptor is passed at the location pointed to by `size`. `_os_gs_cpypd()` also stores the actual number of bytes copied at the location pointed to by `size`.

`path_desc` is a pointer to the buffer for the path descriptor data.

If the `size` value is greater than the size of the target path descriptor, the actual size of the path descriptor is returned.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_cstats()

Get Cache Status Information

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_gs_cstats(
     path_id      path,
     cachestats   *cache_inf);
```

Description

`_os_gs_cstats()` returns a copy of the current `cachestats` structure.

`path` specifies the number of a path open on the device.

`cache_inf` is a pointer to the location where `_os_gs_cstats()` stores the structure containing information about RBF caching.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_devnm()

Return Device Name

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_devnm(
     path_id      path,
     char         *name);
```

Description

`_os_gs_devnm()` returns the name of the device associated with the specified path.

`path` specifies the path number.

`name` is a pointer to the location where `_os_gs_devnm()` stores the device name.

Some networked devices using NFM may require a buffer size of 128 or 256 bytes for the device name.

Note

On OS-9, the buffer pointed to by `name` should be at least 64 bytes to accommodate a device name. On OS-9 for 68K, the buffer pointed to by `name` is a fixed 32 bytes.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_devtyp(
     path_id     path,
     u_int16     *type,
     u_int16     *class);
```

Description

`_os_gs_devtyp()` returns the type and class of the device associated with the specified path number.

`path` specifies the path number.

`type` is a pointer to the location where `_os_gs_devtyp()` stores the device type.

`class` is a pointer to the location where `_os_gs_devtyp()` stores the device class.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_diskfree()

Return Information About RBF Disk Free Space

Syntax

```
error_code _os_gs_diskfree(path_id  path,
                           u_int32  *totblocks,
                           u_int32  *blksize,
                           u_int32  *avail,
                           u_int32  *contig);
```

Description

`_os_gs_diskfree()` returns information about RBF disk free space.

`totblocks` is set to the total number of blocks on the disk.

`blksize` is set to the size of blocks used on the disk (256, 512, etc.)

`avail` is set to the total number of free blocks on the disk.

`contig` is set to the number of blocks in the largest contiguous area.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_dopt()

Read Device Path Options

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_dopt(
     path_id     path,
     u_int32     *size,
      void       *dopts);
```

Description

`_os_gs_dopt()` copies the initial (default) device path options into your buffer. These options are used for initializing new paths to the device.

`path` specifies the path number.

The size of the buffer is passed at the location pointed to by `size`. `_os_gs_dopt()` also stores the actual number of bytes copied at the location pointed to by `size`.

`dopts` is a pointer to a buffer for the device path options.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_dsize()

Get Number of Blocks on Media

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_gs_dsize(
     path_id      path,
     u_int32      *totblocks,
     u_int32      *blocksize);
```

Description

`_os_gs_dsize()` gets information about the number of blocks on the media.

`path` specifies the drive's path number.

`totblocks` is a pointer to the location where `_os_gs_dsize()` stores the total number of blocks on the device.

`blocksize` is a pointer to the location where `_os_gs_dsize()` stores the size of a disk block in bytes.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_edt()

Get I/O Interface Edition Number

Syntax

```
#include <ioedt.h>
#include <types.h>
error_code _os_gs_edt(
     path_id     path,
     u_int32     *edition);
```

Description

`_os_gs_edt()` returns the I/O interface edition number of the driver and validates the compatibility of drivers and file managers.

`path` specifies the driver's path number.

`edition` is a pointer to the location where `_os_gs_edt()` stores the driver I/O interface edition number.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_eof()

Test for End of File

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_eof(path_id path);
```

Description

`_os_gs_eof()` returns the `EOS_EOF` error if the current position of the file pointer associated with the specified path is at the end-of-file. SCF never returns `EOS_EOF` with this `GetStat`.

`path` specifies the driver's path number.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_fd()

Read File Descriptor Sector

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_gs_fd(
     path_id     path,
     u_int32     size,
     fd_stats    *fdbuf);
```

Description

`_os_gs_fd()` returns a copy of the file descriptor sector for the file associated with the specified path.

`path` specifies the driver's path number.

`size` is the number of bytes of the file descriptor to copy.

`fdbuf` is a pointer to the buffer for the file descriptor sector.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_fdaddr()

Get File Descriptor Block Address for Open File

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_gs_fdaddr(
     path_id      path,
     u_int32      *addr);
```

Description

`_os_gs_fdaddr()` returns the file descriptor block number associated with the specified path number.

`path` specifies the driver's path number.

`addr` is a pointer to the location where `_os_gs_fdaddr()` stores the block address of the file descriptor.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_fdinf()

Get Specified File Descriptor Sector

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_gs_fdinf(
     path_id     path,
     u_int32     size,
     u_int32     fdblk,
     fd_stats    *fdbuf);
```

Description

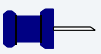
`_os_gs_fdinf()` returns a copy of the specified file descriptor sector for the file associated with the specified file descriptor sector.

`path` specifies an open path on a particular device.

`size` specifies the number of bytes of the file descriptor block to copy.

`fdblk` specifies the file descriptor sector number to get.

`fdbuf` is a pointer to the buffer for the file descriptor block.



Note

Typically, `_os_gs_fdinf()` is used to rapidly scan a directory on a device. You do not need to specify the path number of the file for which you want the file descriptor in `path`. However, `path` must be an open path on the same device as the file. `path` is typically the path number of the directory you are currently scanning.

Note

Only super users may use this call.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os9_gs_free()

Return Amount of Free Space on Device

Syntax

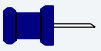
```
#include <sg_codes.h>
#include <types.h>
error_code _os9_gs_free(
     path_id     path,
     u_int32     *free);
```

Description

`_os9_gs_free()` returns the amount of free space on a device.

`path` specifies the path number of the device.

`free` is a pointer to the location where `_os9_gs_free()` stores the amount of free space. The amount of free space is specified in bytes.



Note

`_os9_gs_free()` is only supported by the NRF file manager.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_luopt()

Read Logical Unit Options

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_luopt(
     path_id      path,
     u_int32      *size,
     void         *luopts);
```

Description

`_os_gs_luopt()` copies the contents of the logical unit options for a path into the buffer pointed to by `luopts`.

`path` specifies the driver's path number.

The size of the buffer is passed at the location pointed to by `size`. `_os_gs_luopt()` also stores the actual number of bytes copied at the location pointed to by `size`. `size` may not be less than the size of the file manager's logical unit option section.

`luopts` is a pointer to your buffer.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_parity()

Calculate Parity of File Descriptor

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_gs_parity(
     path_id      path,
     fd_stats     *fdbuf,
     u_int32      *parity);
```

Description

`_os_gs_parity()` uses a 32-bit vertical parity to validate file descriptor structures.

`path` specifies an open path to the file with which the file descriptor is associated.

`fdbuf` is a pointer to the file descriptor block.

`parity` is a pointer to the location where `_os_gs_parity()` stores the resulting parity.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_popt()

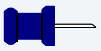
Read Path Descriptor Option Section

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_popt(
     path_id      path,
     u_int32      *size,
     void         *user_opts);
```

Description

`_os_gs_popt()` copies the option section of the path descriptor into the variable-sized area pointed to by `user_opts`. You must include `rbf.h`, `sbf.h`, and/or `scf.h` for the corresponding file managers and declare `size` according to the size of the `rbf_opts`, `sbf_opts`, or `scf_opts`. `path` specifies the driver's path number. The requested number of bytes is passed at the location pointed to by `size`. `_os_gs_popt()` also stores the actual number of bytes copied at the location pointed to by `size`. `user_opts` is a pointer to your buffer.



Note

On OS-9 for 68K, 128 bytes are always allocated by the binding, although only the specified number of bytes are passed back to you.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_pos()

Get Current File Position

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_pos(
     path_id     path,
     u_int32     *position);
```

Description

`_os_gs_pos()` returns the current position of the file pointer associated with the specified path.

`path` specifies the driver's path number.

`position` is a pointer to the location where `_os_gs_pos()` stores the file position.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os9_gspump()

Return Data for Specified Process' Memory Map

Syntax

```
#include <process.h>
#include <types.h>
error_code _os9_gspump(
     process_id   proc_id,
     u_int32      size,
     void         *buffer);
```

Description

`_os9_gspump()` returns data about a specified process' memory map for debugging purposes.

`proc_id` specifies the process ID.

`size` specifies the size of the buffer.

`buffer` is a pointer to the storage buffer.

One word in this format is returned for each memory block in the system. This information is taken from the process' SSM data structure. If the specified address space is not large enough, only the information that fits in the area is returned.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$GSPUMp` ***OS-9 for 68K Technical Manual***

_os_gs_ready()

Test for Data Ready

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_ready(
     path_id     path,
     u_int32     *incount);
```

Description

`_os_gs_ready()` checks for data available to be read on the specified path. RBF devices do not return the `EOS_NOTRDY` error because there is always information available to be read on RBF devices. `_os_gs_ready()` returns the number of bytes left in `incount`.

`path` specifies the driver's path number.

`incount` is a pointer to the location where `_os_gs_ready()` stores the number of characters available to be read.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_size()

Return Current File Size

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_size(
     path_id      path,
     u_int32      *size);
```

Description

`_os_gs_size()` returns the current size of the file associated with the specified path.

`path` specifies the driver's path number.

`size` is a pointer to the location where `_os_gs_size()` stores the file size.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_id(
     process_id   *proc_id,
     u_int16      *priority,
     u_int16      *age,
     int32        *schedule,
     u_int16      *group,
     u_int16      *user);
```

Description

`_os_id()` returns the caller's process ID number, current process priority and age, scheduling constant, and owner ID.

`proc_id` is a pointer to the location where `_os_id()` stores the current process ID number.

`priority` is a pointer to the location where `_os_id()` stores the priority of the current process.

`age` is a pointer to the location where `_os_id()` stores the age of the current process.

`schedule` is a pointer to the location where `_os_id()` stores the scheduling constant of the current process.

`group` is a pointer to the location where `_os_id()` stores the group number of the current process.

`user` is a pointer to the location where `_os_id()` stores the user number of the current process.

Note

On OS-9 for 68K, the binding for this call, performs conversions which allow compatibility with OS-9. If speed is a consideration, see `_os9_id()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os9_id()`

Syntax

```
#include <process.h>
#include <types.h>
error_code _os9_id(
     process_id   *proc_id,
     u_int16      *priority,
     u_int16      *group,
     u_int16      *user);
```

Description

`_os9_id()` returns the caller's process ID number, group and user ID, and current process priority.

`proc_id` is a pointer to the location where `_os9_id()` stores the process ID.

`priority` is a pointer to the location where `_os9_id()` stores the current process priority.

`group` is a pointer to the location where `_os9_id()` stores the group ID.

`user` is a pointer to the location where `_os9_id()` stores the user ID.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_initdata()

Initialize Static Storage from Module

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_initdata(
     mh_com       *mod_head,
     void         *statics);
```

Description

`_os_initdata()` clears the uninitialized data area, copies the module header's initialized data to the specified data area, and clears the remote data area (if it exists). It then adjusts the code and data offsets.

`mod_head` is a pointer to the module header.

`statics` is a pointer to the data area.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <signal.h>
#include <types.h>
#include <cglob.h>
error_code _os_intercept(
     void         (*func)(),
      void        *data_ptr);
```

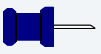
Description

`_os_intercept()` installs a signal intercept routine. The intercept routine is entered asynchronously because a signal may be sent at any time, similar to an interrupt. The signal code is passed as a parameter to the intercept routine.

`func` is a pointer to the intercept routine's address.

`data_ptr` is a pointer to the intercept routine's global storage. It usually contains the address of the program's data area.

`_os_rte()` must be used to exit from the intercept routine (`func`). If the routine fails to use `_os_rte()` at every possible exit, unpredictable results occur.



Note

The intercept routine should be short and fast, such as setting a flag in the process' data area. Avoid complicated system calls (such as I/O).

In OS-9 for 68K, `data_ptr` **must** be `_glob_data` to provide compatibility between OS-9 for 68K and OS-9.

`signal()` and `intercept()` use `_os_intercept()` internally; do not use `_os_intercept()` with either of these functions.

Each time the intercept routine is called, the state of the processor (such as its registers) is pushed on to the user's stack.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`intercept()`	
`_os_rte()`	
`_os_send()`	
`signal()`	
`F$Icpt`	***OS-9 for 68K Technical Manual***
`F_ICPT`	***OS-9 Technical Manual***

Syntax

```
#include <modes.h>
error_code _os_ioconfig(
     u_int32     code,
     void        *param);
```

Description

`_os_ioconfig()` is a wildcard call used to configure elements of the I/O subsystem which may or may not be associated with an existing path. It is intended to be used to dynamically reconfigure system I/O resources at runtime. The target I/O resources may be system-wide resources or they may be process- or path-specific, depending on the nature of the configuration call being made.

`code` is the IO configuration command code. Refer to the ***OS-9 Technical Manual*** for a list of valid command codes for your version of the operating system.

`param` is a pointer to any additional parameters required by the specified configuration function.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

See Also

`_os_config()`

_os_iodel()

Check for Use of I/O Module

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_iodel(mh_com *mod_head);
```

Description

`_os_iodel()` is executed when the kernel unlinks a file manager, device driver, or device descriptor module. It is used to determine if the I/O system is still using the module.

`mod_head` is a pointer to the module header.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ioexit()

Terminate I/O for Exiting Process

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_ioexit(
     process_id   proc_id,
     u_int32      path_cnt);
```

Description

`_os_ioexit()` is executed when the kernel terminates or chains to a process.

`proc_id` specifies the process ID.

`path_cnt` specifies the number of I/O paths.

If the most significant bit of `path_cnt` is cleared, the process' default data and execution directory paths and all other open paths in the path translation table are closed. The I/O process descriptor is also deallocated.

If the most significant bit of `path_cnt` is set, the remaining bits specify the number of paths to leave open. The default directory paths are not closed, and the I/O process descriptor is not de-allocated.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_iofork()

Set Up I/O for New Process

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_iofork(
     process_id   par_proc_id,
     process_id   new_proc_id,
     u_int32      path_cnt);
```

Description

`_os_iofork()` is executed when the kernel creates a new process. `_os_iofork()` creates an I/O process descriptor for the new process.

`par_proc_id` is the parent's process ID.

`new_proc_id` is the process ID of the new process.

`path_cnt` is the number of I/O paths.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os9_ioqueue()

Enter I/O Queue

Syntax

```
#include <process.h>
#include <types.h>
error_code _os9_ioqueue(process_id proc_id);
```

Description

`_os9_ioqueue()` links the calling process into the I/O queue and performs an untimed sleep.

`proc_id` specifies the process ID of the process currently in control of the device.

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_findpd()`
`_os_send()`
`_os_sleep()`
`_os9_sleep()`
`F$IOQu`

OS-9 for 68K Technical Manual

_os_irq()

Add or Remove Device from IRQ Table

Syntax

```
#include <regs.h>
#include <types.h>
error_code _os_irq(
     u_int32     vector,
     u_int32     priority,
     void        *irq_entry,
     void        *statics);
```

Description

`_os_irq()` installs an IRQ service routine into the system polling table.

`vector` specifies the vector number. Only the lower 16 bits of `vector` are used.

`priority` specifies the priority. Only the lower 16 bits of `priority` are used.

- If `priority` is 0, the routine is polled first on the vector.
- If `priority` is 1, the routine is polled first and no other device can have a priority of 1 on the vector.
- If `priority` is 255, the routine is polled last on the vector.

`irq_entry` is a pointer to the IRQ service routine entry point. If `irq_entry` is 0, the call deletes the IRQ service routine.

`statics` is a pointer to the global static storage. `statics` must be unique to the device.

WARNING

The called `irq` function must not be compiled with `stackcheck` because the stack is out of range at this time.

For More Information

Refer to the ***OS-9 Technical I/O Manual*** for more information.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F_IRQ` ***OS-9 Technical Manual***

_os9_irq()

Add or Remove Device from IRQ Table

Syntax

```
#include <regs.h>
#include <types.h>
error_code _os9_irq(
     u_int32     vector,
     u_int32     priority,
     void        *irq_entry,
     void        *statics,
     void        *port);
```

Description

`_os9_irq()` installs an IRQ service routine into the system polling table.

`vector` specifies the vector number. Only the lower 16 bits of `vector` are used.

`priority` specifies the priority. Only the lower 16 bits of `priority` are used.

- If `priority` is `0`, the routine is polled first on the vector.
- If `priority` is `255`, the routine is polled last on the vector.

`irq_entry is` a pointer to the IRQ service routine entry point. If `irq_entry` is `0`, the call deletes the IRQ service routine.

`statics` is a pointer to the global static storage. `statics` must be unique to the device.

`port` is the port address.

For More Information

Refer to the ***OS-9 for 68K Technical I/O Manual*** for more information.

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$IRQ`	***OS-9 for 68K Technical Manual***

_os_julian()

Get Julian Date

Syntax

```
#include <time.h>
#include <types.h>
error_code _os_julian(
     u_int32     *time,
     u_int32     *date);
```

Description

`_os_julian()` converts Gregorian dates to Julian dates.

`time` is a pointer to the location where the time in the form `00hhmmss` is passed. `_os_julian()` stores the time in seconds since midnight in the location pointed to by `time`.

`date` is a pointer to the location where the date in the form `yyyymmdd` is passed. `_os_julian()` stores the Julian date in the location pointed to by date.

For More Information

Refer to your operating system technical manual for more information.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <types.h>
int os9kexec(
     u_int32      (*func)(),
     u_int16      priority,
     u_int16      pathcnt,
     u_char       *modname,
     u_char       **argv,
     u_char       **envp,
     u_int32      datasize,
     void         *extra1,
     void         *extra2,
     void         *extra3,
     void         *extra4);
```

Description

`os9kexec()` is a high-level fork/chain interface that prepares the parameter and environment list before a process is created.

`os9kexec()` closely resembles the UNIX `execve()` system call.

`func` is a pointer to one of the following functions that create a new process: `_os_fork()`, `_os_forkm()`, `_os_chain()`, `_os_chainm()`, `_os_dfork()`, and `_os_dforkm()`. `func` passes information to a new process as binary data specified by a pointer and size. It is up to the forked process to interpret the data.

`priority` is the priority at which the new process is to run. If `priority` is zero, the new process receives the priority of the calling process.

`pathcnt` is the number of open paths to pass to the new process. Normally, `pathcnt` is 3, causing the 3 standard paths to be passed. A `pathcnt` of zero passes **no** open paths.

modname is a pointer to a string naming the new primary module or to a module in the case of the functions ending in m (_os_forkm(), _os_chainm(), and _os_dforkm()).

argv is a pointer to the parameter pointer list that the new process receives. By convention, argv[0] is the name of the new module. A null pointer marks the end of the argv list.

envp is a pointer to the environment pointer list that the new process receives. It points to environment variables. A null pointer marks the end of the envp list.

datasize specifies the additional memory (in bytes) to allocate to the new process' stack memory. If datasize is zero, the default stack size required by the process is used.

The first 7 parameters are identical for _os_fork(), _os_forkm(), _os_chain(), _os_chainm(), _os_dfork(), and _os_dforkm().

The remaining four parameters have different meanings, depending on the function specified by func:

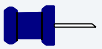
Table 2-20 os9kexec() Additional Parameters

Function	Parameter	Definition
_os_fork()	extra1	The child process ID returns to where extra1 points
	extra2	Type/language of the new process
	extra3	Orphan: 0 = Normal child process 1 = Process not associated with parent
_os_forkm()	extra1	The child process ID returns to where extra1 points
	extra2	Orphan: 0 = Normal child process 1 = Process not associated with parent

Table 2-20 os9kexec() Additional Parameters (continued)

Function	Parameter	Definition
_os_chain()	extra1	Type/language of the new process
_os_chainm()		Does not use the extra parameters
_os_dfork()	extra1	The child process ID returns to where extra1 points
	extra2	Type/language of the new process
	extra3	Pointer to register stack for child process
	extra4	Pointer to floating point register stack for child process
_os_dforkm()	extra1	The child process ID returns to where extra1 points
	extra2	Pointer to register stack for child process
	extra3	Pointer to floating point register stack for child process

If func indicates a fork function and the initial attempt to create the child process fails due to EOS_PNNF (path name not found), os9kexec() calls modloadp() with modname to load the module from disk and attempts the fork again. This action is the same as the shell handling of executable files.

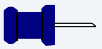


Note

If you use `os9kexec()` when `func` is `_os_chain()` or `_os_chainc()`, you must either:

- Ensure that the module is in memory or in your current execution directory (see `modlink()`, `modload()`, and `modloadp()`).
- Provide a full pathlist to the module.

`os9kexec()` returns the value of `(*func)()`. It returns `0` if successful; otherwise, it returns an error number.



Note

Any program that executes a C program should do so using the `os9kexec()` interface. The `cstart` module can handle parameter strings passed by `os9fork()` and `chain()`, but no environment is available and the `argv` pointer list is separated by white space.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

For More Information

The primary purpose of the `sys_clib.l` library is to provide a library for compatibility with the Microware K&R C compiler. For additional information, refer to **Appendix A: Prototyping sys_clib.l Functions**.

See Also

`_os_exec()`
`_os_fork()`
`_os_forkm()`
`_os_chain()`
`_os_chainm()`
`_os_dfork()`
`_os_dforkm()`

Example

```
#include <module.h>
#include <types.h>
#include <const.h>
extern  int    _os_fork();
extern  char   **_environ;

char   * argblk[] = {
       "rename",
       "oldname",
       "newname",
       "-x",
       0,
}

main()
{
  u_int16  type_lang = mktypelang (MT_PROGRAM,
                                   ML_OBJECT);
  process_id   pid;

  If ((errno = os9kexec (_os_fork,
     0,            * no change in priority
     3,            * pass stdin, stdout, and stderr
     angblk[0],    * pointer to module name
     argblk,       * the argument list
     _environ,     * environment list for new process
     0,            * no extra memory
     &pid          * pointer to the process ID
     type_lang,
     0))           * normal child process

     != SUCCESS)
  printf ("can't fork %s\n", argblk[0]);
  else
     wait(0)
}
```

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_link(
     char         **mod_name,
     mh_com       **mod_head,
     void         **mod_exec,
     u_int16      *type_lang,
     u_int16      *attr_rev);
```

Description

`_os_link()` searches the current module directory (and an alternate module directory in OS-9) for a module whose name, type, and language match the parameters.

The module name is passed at the location pointed to by `mod_name`. On OS-9, if `mod_name` is an explicit module directory pathlist (for example, `/usr/pers1/prog`), `_os_link()` also stores the pointer to the module that was successfully linked (for example, `prog`) at the location pointed to by `mod_name`.

`mod_head` is a pointer to the location where `_os_link()` stores the address of the module's header.

`mod_exec` is a pointer to the location where `_os_link()` stores the absolute address of the module's execution entry point.

The type and language of the module are passed at the location pointed to by `type_lang`. `_os_link()` also stores the actual type and language at the location pointed to by `type_lang`. If `type_lang` is zero (specifying any), any module can be linked to regardless of the type and language.

`attr_rev` is a pointer to the location where `_os_link()` stores the attribute and revision level of the module.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_load()`
`_os_unlink()`
`_os_unload()`

`F$Link`	***OS-9 for 68K Technical Manual***
`F_LINK`	***OS-9 Technical Manual***

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_linkm(
     mh_com       *mod_head,
     void         **mod_exec,
     u_int16      *type_lang,
     u_int16      *attr_rev);
```

Description

`_os_linkm()` causes OS-9 to link to the module specified by `mod_head`.

`mod_head` is a pointer to the module.

`mod_exec` is a pointer to the location where `_os_linkm()` stores the absolute address of the module's execution entry point.

`attr_rev` is a pointer to the location where `_os_linkm()` stores the attribute and revision level of the module.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_load()`
`_os_unlink()`
`_os_unload()`

_os_load()

Load Module(s) from File

Syntax

```
#include <module.h>
#include <modes.h>
#include <memory.h>
#include <types.h>
error_code _os_load(
     char          *mod_name,
     mh_com        **mod_head,
     void          **mod_exec,
     u_int32       mode,
     u_int16       *type_lang,
     u_int16       *attr_rev,
     u_int32       color);
```

Description

`_os_load()` opens a file specified by the pathlist. It reads one or more memory modules from the file into memory until an error or end of file is reached. Then, it closes the file. Modules are loaded into the highest physical memory available of the specified `color` type.

`mod_name` is a pointer to the module name.

`mod_head` is a pointer to the location where `_os_load()` stores the pointer to the module.

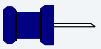
`mod_exec` is a pointer to the location where `_os_load()` stores the pointer to the module execution entry point.

`mode` specifies the access mode. Only the lower 16 bits of `mode` are used.

The type and language of the module are passed at the location pointed to by `type_lang`. `_os_load()` also stores the actual type and language at the location pointed to by `type_lang`. If `type_lang` is zero, any module can be linked to regardless of the type and language.

`attr_rev` is a pointer to the location where `_os_load()` stores the attribute and revision level of the module.

When a module is loaded, its name is added to the system module directory, and the first module read is linked. The returned parameters are the same as those returned by a link call and apply only to the first module loaded.



Note

`_os_load()` does not work on SCF devices**.**

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Load`	***OS-9 for 68K Technical Manual***
`F_LOAD`	***OS-9 Technical Manual***

_os_loadp()

Load and Link to Memory Module Using PATH

Syntax

```
#include <module.h>
#include <modes.h>
#include <types.h>
error_code _os_loadp(
     char         *name,
     u_int32      mode,
     char         *nameptr,
     mh_com       **mod_head);
```

Description

`_os_loadp()` opens a file specified by the pathlist. It reads one or more memory modules from the file into memory until an error or end of file is reached; then it closes the file.

The PATH environment variable determines the alternate directories to search for the named file.

`name` is a pointer to the file name.

`mode` specifies the access mode. Only the lower 16 bits of `mode` are used.

`nameptr` is a pointer to a buffer for the pathlist of the successfully loaded file. Use NULL to indicate that no complete pathlist is required.

`mod_head` is a pointer to the location where `_os_loadp()` stores the pointer to the module.

For More Information

Refer to your operating system technical manual for information regarding the PATH environment variable.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_load()`
`_os_exec()`

_os_makdir()

Make New Directory

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_makdir(
     char         *name,
     u_int32      mode,
     u_int32      perm, ...);
```

Description

`_os_makdir()` creates and initializes a new directory as specified by the pathlist. `_os_makdir()` is the only way to create a new directory file. `_os_makdir()` fails on non-multifile devices.

`name` is a pointer to the pathlist.

`mode` specifies the access mode. Only the lower 16 bits of `mode` are used.

`perm` specifies the access permissions.

A variable parameter may be included to specify the initial allocation size.

`_os_makdir()` does not return a path number. You should use `_os_open()` to open a directory. The new directory automatically has its directory bit set in the access permission attributes. The remaining attributes are specified by the bytes passed in `mode` and `perm`.

For More Information

Refer to your operating system technical manual for the available modes and attributes.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_makmdir()

Create New Module Directory

Syntax

```
#include <moddir.h>
#include <types.h>
error_code _os_makmdir(
     char         *name,
     u_int32      perm);
```

Description

`_os_makmdir()` creates a new module directory. The name of the new module directory is relative to the current module directory.

`name` is a pointer to the name of the new module directory.

`perm` specifies the access permissions for the new module directory. Only the lower 16 bits of `perm` are used.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <memory.h>
#include <types.h>
error_code _os_mem(
     u_int32     *size,
     void        **mem_ptr);
```

Description

`_os_mem()` contracts or expands the process' data memory area. The requested size is rounded up to an even memory allocation block. Additional memory is allocated contiguously upward (towards higher addresses) or deallocated downward from the old highest address.

The desired memory size in bytes is passed at the location pointed to by `size`. `_os_mem()` also stores the actual size of the memory at the location pointed to by `size`. If the integer pointed to by `size` is zero, `_os_mem()` treats the call as a request for information and returns the current upper bound in `mem_ptr` and the amount of free memory in `size`. Otherwise, `mem_ptr` is a pointer to the location where `_os_mem()` stores the pointer to the new end of data segment plus 1.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Mem`	***OS-9 for 68K Technical Manual***
`F_MEM`	***OS-9 Technical Manual***

_os_mkmodule()

Create Data Module of Specified Color Type

Syntax

```
#include <module.h>
#include <memory.h>
#include <types.h>
error_code _os_mkmodule(
     char         *mod_name,
     u_int32      size,
     u_int16      *attr_rev,
     u_int16      *type_lang,
     u_int32      perm,
     void         **mod_exec,
     mh_com       **mod_head,
     u_int32      color);
```

Description

`_os_mkmodule()` creates a memory module with the specified name and attributes in the specified memory. The module is created with a valid CRC and entered into the system module directory. You can specify the type/language of the module and color of memory in which to make the module.

`mod_name` is a pointer to the name of the data module.

`size` specifies the size of the data portion to clear.

`attr_rev` is a pointer to the location where `_os_mkmodule()` stores the module's attribute and revision.

The type and language for the module are passed at the location pointed to by `type_lang`. `_os_mkmodule()` also stores the actual type and language at the location pointed to by `type_lang`.

`perm` specifies the access permissions for the module. Only the lower 16 bits of `perm` are used.

`mod_exec` is a pointer to the location where `_os_mkmodule()` stores the pointer to the module data.

`mod_head` is a pointer to the location where `_os_mkmodule()` stores the pointer to the module header.

`color` is the memory color type: `SYSRAM`, `VIDEO1`, or `VIDEO2`. If `color` is zero, no memory type is specified. Consequently, the module is made in whatever memory the system allocates.

Note

The created module always begins at offset $34 within the module. This implies that this call cannot conveniently make program modules, trap handlers, file managers, device drivers, and device descriptors.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_datmod()`
`_os_setcrc()`
`_os_move()`

_os_modaddr()

Find Module Given Pointer

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_modaddr(
     void         *mem_ptr,
     mh_com       **mod_head);
```

Description

`_os_modaddr()` locates a module given a pointer to any position within the module and returns a pointer to the module's header.

`mem_ptr` is a pointer to any position within the module.

`mod_head` is a pointer to the location where `_os_modaddr()` stores the pointer to the module header.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <memory.h>
#include <types.h>
error_code _os_move(
     void         *from,
     void         *to,
     u_int32      count);
```

Description

`_os_move()` is a fast “block-move” subroutine which copies data bytes from one address space to another, usually from system to user or vice versa. This call allows the source and destination buffers to overlap.

`from` is a pointer to the source data.

`to` is a pointer to the destination data.

`count` is the byte count to copy.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_nproc()

Start Next Process

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_nproc(void);
```

Description

`_os_nproc()` removes the next process from the active process queue and initiates its execution. `_os_nproc()` does not return to the caller.

Note

The process calling `_os_nproc()` should already be in one of the system's process queues. If not, the process becomes unknown to the system. This occurs even though the process descriptor still exists and is printed out by a `procs` command.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_aproc()`

_os_open()

Open Path to File or Device

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_open(
     const char   *name,
     u_int32      mode,
     path_id      *path);
```

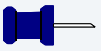
Description

`_os_open()` opens a path to an existing file or device as specified by the pathlist. `_os_open()` returns a path number which is used in subsequent service requests to identify the path. If the file does not exist, an error is returned.

`name` is a pointer to the path name of the existing file or device.

`mode` specifies which subsequent read and/or write operations are permitted. Only the lower 16 bits of `mode` are used.

`path` is a pointer to the location where `_os_open()` stores the resulting path number.



Note

A non-directory file may be opened with no `mode` bits set. This allows you to examine characteristics such as the attributes and size by the `_os_getstat()` system requests, but does not permit any actual I/O on the path.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_attach()`
`_os_close()`
`_os_create()`
`I$Open` ***OS-9 for 68K Technical Manual***
`I_OPEN` ***OS-9 Technical Manual***

Syntax

```
#include <process.h>
#include <types.h>
error_code _os9_panic(
     u_int32     panic_code);
```

Description

`_os9_panic()` is called when the kernel detects a disastrous, but not necessarily fatal, system condition. Ordinarily, `_os9_panic()` is undefined and the system dies.

`panic_code` specifies the panic code to output.

`_os9_panic()` is called only when the kernel believes there are no processes remaining to be executed. Although the system is probably dead at this point, it may not be. Interrupt service routines or system-state alarms could cause the system to become active.

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os9_setsvc()`
`F$Panic` ***OS-9 for 68K Technical Manual***

_os_permit()

Allow Access to Memory Block

Syntax

```
#include <process.h>
#include <types.h>
#include <modes.h>
error_code _os_permit(void *mem_ptr, u_int32 size,
                      u_int32 perm, process_id pid);
```

Description

`_os_permit()` allows a process to access a block of memory. The `_os_permit()` service routine must update SSM (System Security Module) data structures to show that a process may access the specified memory in the requested mode.

Note

Only super-group users may use the `_os_permit()` function.

`mem_ptr` is a pointer to the beginning of the memory area to grant access permissions.

`size` is the size of the memory area.

`perm` is the permissions to grant. Only the lower 16 bits of `perm` are used.

`pid` is the process identifier of the target process (not used for OS-9 for 68K).

Note

`_os_permit()` only performs a useful function on MMU/SSM systems. On non-MMU/SSM systems, `_os_permit()` simply returns without allowing access to the memory block.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Permit`	***OS-9 for 68K Technical Manual***
`F_PERMIT`	***OS-9 Technical Manual***

Syntax

```
#include <errno.h>
#include <types.h>
error_code _os_perr(
     path_id      path,
     u_int32      errnum);
```

Description

`_os_perr()` is the system's error reporting facility. It writes an error message to the standard error path. Most OS-9 for 68K systems print `ERROR #mmm.nnn`. Error numbers `000:000` to `063:255` are reserved for the operating system.

`path` is the path to the error message file. If `path` is zero, standard format is specified.

`errnum` is the error number. Only the lower 16 bits of `errnum` are used.

If an error path number is specified, the path is searched for a text description of the error encountered. The error message path contains an ASCII file of error messages. Each line may be up to 80 characters long. If the error number matches the first 7 characters in a line (that is, `000:215`), the rest of the line is printed along with the error number.

Error messages may be continued on several lines by beginning each continuation line with a space. An example error file might contain lines like this:

```
000:214 (E$FNA) File not accessible.
An attempt to open a file failed.  The file was found, but is
inaccessible to you in the requested mode.  Check the file's
owner ID and access attributes.

000:215 (E$BPNam) Bad pathlist specified.
The specified pathlist is syntactically incorrect.
```

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_protect()

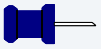
Prevent Access to Memory Block

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_protect(
     void         *mem_ptr,
     u_int32      size,
     u_int32      perm,
     process_id   pid);
```

Description

`_os_protect()` removes a process' permission to access a block of memory.



Note

Only super-group users may use the `_os_protect` function.

`mem_ptr` is a pointer to the beginning of the memory area to remove access permissions.

`size` specifies the size of memory.

`perm` specifies the permissions to remove. Only the lower 16 bits of `perm` are used.

`pid` specifies the process identifier for the target process.

Note

`_os_protect()` only performs a useful function on MMU/SSM systems. On non-MMU/SSM systems, `_os_protect()` simply returns without preventing access to the memory block.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Protect`	***OS-9 for 68K Technical Manual***
`F_PROTECT`	***OS-9 Technical Manual***

_os_prsnam()

Parse Path Name

Syntax

```
#include <strings.h>
#include <types.h>
error_code _os_prsnam(
     char         *name,
     u_int32      *length,
     char         *delimiter,
     char         **updated);
```

Description

`_os_prsnam()` parses a string for a valid pathlist element and returns its size. This call parses one element in a pathlist, not the entire pathlist. A valid pathlist element may contain the following characters:

`A - Z`	Upper case letters	`.`	Periods
`a - z`	Lower case letters	`_`	Underscores
`0 - 9`	Numbers	`$`	Dollar signs

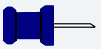
Other characters terminate the name and are returned as the pathlist delimiter.

`name` is a pointer to the pathlist string.

`length` is a pointer to the location where `_os_prsnam()` stores the length of the pathlist element.

`delimiter` is a pointer to the location where `_os_prsnam()` stores the pathlist delimiter.

`updated` is a pointer to the location where `_os_prsnam()` stores the pointer to the first character of `name`.



Note

Several `_os_prsnam()` calls are needed to process a pathlist with more than one name. `_os_prsnam()` terminates a name when it reaches a delimiter character. Usually, pathlists must be terminated with a null byte.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_cmpnam()`

Syntax

```
#include <io.h>
#include <types.h>
error_code _os_rdalst(
     char         *buffer,
     u_int32      *count);
```

Description

`_os_rdalst()` copies the system alias list to the caller's buffer. At most, `count` bytes are copied to the buffer. Each alias entry is null terminated.

`buffer` is a pointer to the buffer into which to copy the alias list.

The number of total bytes to copy is passed at the location pointed to by `count`. `_os_rdalst()` also stores the actual number of bytes copied at the location pointed to by `count`.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_read()

Read Data from File or Device

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_read(
     path_id      path,
     void         *buffer,
     u_int32      *count);
```

Description

`_os_read()` reads the number of bytes from the specified path. The path must previously have been opened in read or update mode. The data is returned exactly as read from the file or device without additional processing or editing such as backspace and line delete. If not enough data is in the file to satisfy the read request, fewer bytes are read than requested, but an end-of-file error is not returned until the next `_os_read()` service request returns an end-of-file error. `path` specifies the path number. `buffer` is a pointer to the caller's data buffer.

The number of bytes requested to be read is passed at the location pointed to by `count`. `_os_read()` also stores the number of bytes actually read at the location pointed to by `count`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_readln()`	
`I$Read`	***OS-9 for 68K Technical Manual***
`I_READ`	***OS-9 Technical Manual***

_os_readln()

Read Text Line with Editing

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_readln(
     path_id      path,
     void         *buffer,
     u_int32      *count);
```

Description

`_os_readln()` reads the specified number of bytes from the input file or device until an end-of-line character is encountered. On SCF-type devices, `_os_readln()` also causes line editing such as backspacing, line delete, echo, and automatic line feed to occur. `path` specifies the path number. `buffer` is a pointer to the location where the bytes to be read are placed.

The number of bytes requested is passed at the location pointed to by count. `_os_readln()` also stores the number of bytes actually read at the location pointed to by count.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_read()`

`I$ReadLn`	***OS-9 for 68K Technical Manual***
`I_READLN`	***OS-9 Technical Manual***

Syntax

```
#include <lock.h>
#include <types.h>
error_code _os_rellk(
     lk_desc     *lock);
```

Description

`_os_rellk()` releases ownership of a resource lock and activates the next process waiting to acquire the lock. The next process in the lock's queue is activated and granted exclusive ownership of the resource lock. If no other process is waiting on the lock, the lock is simply marked free for acquisition.

`lock` is a pointer to the `lk_desc` structure of the lock to release.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`lock.l`

See Also

`_os_acqlk()`
`_os_caqlk()`
`_os_crlk()`
`_os_ddlk()`
`_os_dellk()`
`_os_waitlk()`
`F_RELLK`

OS-9 Technical Manual

_os9_retpd()

Return Fixed Block of Memory

Syntax

```
#include <process.h>
#include <types.h>
error_code _os9_retpd(
     u_int32      number,
     void         *table);
```

Description

`_os9_retpd()` deallocates a fixed block of memory.

`number` is the number of the table element. Only the lower 16 bits of number are used.

`table` is a pointer to the address of the table to use.

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os9_allpd()`
`_os_findpd()`

_os_rte()

Return from Signal Interrupt Routine

Syntax

```
#include <signal.h>
#include <types.h>
error_code _os_rte(void);
```

Description

`_os_rte()` terminates a process' signal intercept routine and continues executing the main program.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_intercept()`

F$RTE	***OS-9 for 68K Technical Manual***
F_RTE	***OS-9 Technical Manual***

_os_rtnprc()

Deallocate Process Descriptor

Syntax

```
#include <process.h
#include <types.h
error_code _os_rtnprc(
     process_id   proc_id);
```

Description

`_os_rtnprc()` deallocates a process descriptor previously allocated by `_os_alocproc()`. You must ensure that the process' system resources are returned before calling `_os_rtnprc()`.

`proc_id` identifies the process descriptor.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_alocproc()`
`_os_findpd()`

Syntax

```
#include <regs.h>
#include <alarm.h>
#include <types.h>
error_code _os9_salarm_atdate(
     alarm_id     *alrm_id,
     u_int32      time,
     u_int32      date,
     REGISTERS    *regs);
```

Description

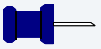
`_os9_salarm_atdate()` executes a system-state subroutine at a specific date and time. The alarm subroutine executes anytime the system date/time becomes greater than or equal to the alarm time.

`alrm_id` is a pointer to the location where `_os9_salarm_atdate()` stores the alarm ID.

`time` is the time for the alarm to go off in the form `00hhmmss`.

`date` is the date for the alarm to go off in the form `yyyymmdd`.

`regs` is a pointer to the register image to pass to the function. The address of the system-state subroutine to execute should be set in `regs->pc`.



Note

`_os9_salarm_atdate()` only allows you to specify the time to the nearest second. However, it does adjust if the system's date and time have changed (using `_os9_setime()`).

Attributes

Operating System:	OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os9_setime()`
`F_ALARM`
OS-9 for 68K Technical Manual

_os_salarm_atime()

Execute System-State Subroutine at Specified Time

Syntax

```
#include <alarm.h>
error_code _os_salarm_atime(
     salarm_id    *alrm_id,
     u_int32      time,
     u_int32      flags,
     u_int32      (*func)(),
     void         *func_pb);
```

Description

`_os_salarm_atime()` executes the specified subroutine after the specified time interval has elapsed. The alarm time is an absolute time, not a relative time. The time value is considered to be a value in seconds since a date specified in Greenwich Mean Time. To determine the specified date, reference the applicable manual, ***OS-9 for 68K Technical Manual*** or ***OS-9 Technical Manual***.

`alrm_id` is a pointer to the location where `_os_salarm_atime()` stores the alarm ID. `time` specifies the time when the alarm expires. `flags` specifies the alarm flags. `func` is a pointer to a function to execute. `func_pb` is a pointer to the function's parameter block.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_salarm_set()`
`F_ALARM` ***OS-9 Technical Manual***

_os9_salarm_atjul()

Execute System-State Subroutine at Julian Date/Time

Syntax

```
#include <regs.h>
#include <alarm.h>
#include <types.h>
error_code _os9_salarm_atjul(
     alarm_id     *alrm_id,
     u_int32      time,
     u_int32      date,
     REGISTERS    *regs);
```

Description

`_os9_salarm_atjul()` executes a system-state subroutine at a specific Julian date and time. The alarm subroutine is executed anytime the system date/time becomes greater than or equal to the alarm time.

`alrm_id` is a pointer to the location where `_os9_salarm_atjul()` stores the alarm ID.

`time` is the time for the alarm to go off in seconds after midnight.

`date` is the Julian day number for the alarm to go off.

`regs` is a pointer to the register image to pass to the function. The address of the system-state subroutine to execute should be set in `regs->pc`.

Note

`_os9_salarm_atjul()` only allows the time to be specified to the nearest second. However, it does adjust if the system's date and time have changed (using `_os9_setime()`).

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os9_setime()`
`F$Alarm`
OS-9 for 68K Technical Manual

_os_salarm_cycle()

Set Cyclic Alarm Clock

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os_salarm_cycle(
     alarm_id     *alrm_id,
     u_int32      time,
     u_int32      flags,
     u_int32      (*func)(),
     void         *func_pb);
```

Description

`_os_salarm_cycle()` executes the specified subroutine after the specified time interval has elapsed. Then the alarm is reset to provide recurring periodic execution of the specified subroutine.

`alrm_id` is a pointer to the location where `_os_salarm_cycle()` stores the alarm ID.

`time` specifies the alarm's time interval. `time` may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second. Usually, the minimum time interval allowed is 2 system clock ticks.

`flags` specifies the alarm flags.

`func` is a pointer to a function to execute.

`func_pb` is a pointer to the function's parameter block.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

_os_salarm_set()
F_ALARM *OS-9 Technical Manual*

_os9_salarm_cycle()

Execute System-State Subroutine Every N Ticks/Seconds

Syntax

```
#include <regs.h>
#include <alarm.h>
#include <types.>h
error_code _os9_salarm_cycle(
     alarm_id     *alrm_id,
     u_int32      time,
     REGISTERS    *regs);
```

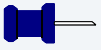
Description

`_os9_salarm_cycle()` executes a system-state subroutine after the specified time interval has elapsed. Then the alarm is reset to provide recurring periodic execution of the system-state subroutine.

`alrm_id` is a pointer to the location where `_os9_salarm_cycle()` stores the alarm ID.

`time` specifies the alarm's time interval. time may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second. Usually, the minimum time interval allowed is 2 system clock ticks.

`regs` is a pointer to the register image to pass to the function. The address of the system-state subroutine to execute should be set in `regs->pc`.



Note

Keep cyclic system-state alarms as fast as possible. Schedule them with as long a cycle as possible to avoid consuming a large portion of available CPU time.

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os9_salarm_set()`
`F$Alarm` ***OS-9 for 68K Technical Manual***

_os_salarm_delete()

Remove Pending Alarm Request

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os_salarm_delete(alarm_id alrm_id);
```

Description

`_os_salarm_delete()` removes a cyclic alarm or any alarm that has not expired.

`alrm_id` specifies the alarm ID. If `alrm_id` is 0 (zero), all pending alarm requests are removed, which are owned by the system process (forked with `TH_SPOWN`).

In a threaded environment, the `alarm_id` is checked to determine the owner of the thread. If a 0 is passed for the `alarm_id`, all pending alarm requests for the system will be removed, which are owned by the system process (forked with `TH_SPOWN`).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Alarm`	***OS-9 for 68K Technical Manual***
`F_ALARM`	***OS-9 Technical Manual***

_os_salarm_delete_sp()

Delete Installed Alarms

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os_salarm_delete_sp(alrm_id, flags);
```

Description

`_os_salarm_delete_sp()` takes a flag value to allow correct deletion of installed alarms, thus providing a workaround for a historic OS-9 bug in `_os_salarm_delete`.

`alrm_id` specifies the alarm ID. If `alrm_id` is zero, all pending alarm requests are removed.

`flags` specifies that the alarm(s) to be deleted are owned by either the system process (`TH_SPOWN`) or by the current user process (`flags=0`).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Alarm`	***OS-9 for 68K Technical Manual***
`F_ALARM`	***OS-9 Technical Manual***

_os_salarm_reset()

Reset Alarm Clock

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os_salarm_reset(
     alarm_id     alrm_id,
     u_int32      time,
     u_int32      flags,
     u_int32      (*func)(),
     void         *func_pb);
```

Description

`_os_salarm_reset()` allows you to change the parameters on an existing alarm. `alrm_id` specifies the alarm ID to reset. `time` specifies the alarm's time interval or a time. A time interval may be specified in system clock ticks or 256ths of a second. The minimum time interval allowed is 2 system clock ticks. `flags` specifies the alarm flags. `func` is a pointer to a function to execute. `func_pb` is a pointer to the function's parameter block.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_salarm_set()`
`F_ALARM` ***OS-9 Technical Manual***

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os_salarm_set(
     alarm_id     *alrm_id,
     u_int32      time,
     u_int32      flags,
     u_int32      (*func)(),
     void         *func_pb);
```

Description

`_os_salarm_set()` executes the specified subroutine after the specified time interval has elapsed.

`alrm_id` is a pointer to the location where `_os_salarm_set()` stores the alarm ID. `time` specifies the alarm's time interval or a time. `time` may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second. Usually, the minimum time interval allowed is 2 system clock ticks. `flags` specifies the alarm flags. `func` is a pointer to the function to execute. `func_pb` is a pointer to the function's parameter block.

Attributes

Operating System:	OS-9K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F_ALARM` ***OS-9 Technical Manual***

_os9_salarm_set()

Execute System-State Subroutine after Specified Time Interval

Syntax

```
#include <regs.h>
#include <alarm.h>
#include <types.h>
error_code _os9_salarm_set(
     alarm_id     *alrm_id,
     u_int32      time,
     REGISTERS    *regs);
```

Description

`_os9_salarm_set()` executes a system-state subroutine after the specified time interval has elapsed.

`alarm_id` is a pointer to the location where `_os9_salarm_set()` stores the alarm ID.

`time` specifies the time interval. The time interval, `time`, may be specified in system clock ticks or 256ths of a second. The minimum time interval allowed is 2 system clock ticks.

`regs` is a pointer to the register image to pass to the function. The address of the system-state subroutine to execute should be set in `regs->pc`.

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Alarm` ***OS-9 for 68K Technical Manual***

Syntax

```
#include <regs.h>
#include <cache.h>
#include <types.h>
error_code _os_scache(
     u_int32     control,
     void        (**cctl)(),
     void        **cctl_data,
     void        *addr,
     u_int32     size);
```

Description

`_os_scache()` performs operations on the system instruction and/or data caches, if there are any. `control` specifies the cache operation. If `control` is zero, the system instruction and data caches are flushed. Non-super group, user-state processes may perform this operation.

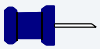
The following table shows the bits that are defined in the `control` parameter for precise operation. Only system-state and super-group processes can use the `control` parameter for precise operation.

Table 2-21 `control` Parameter Bits Defined for _os_scache()

Bit	Name	Description
Bit 0	C_ENDATA	If set, enables the data cache.
Bit 1	C_DISDATA	If set, disables the data cache.
Bit 2	C_FLDATA	If set, flushes the data cache.

Table 2-21 `control` Parameter Bits Defined for _os_scache()

Bit	Name	Description
Bit 3	`C_INVDATA`	If set, invalidates the data cache (if supported by hardware).
Bit 4	`C_ENINST`	If set, enables the instruction cache.
Bit 5	`C_DISINST`	If set, disables the instruction cache
Bit 6	`C_FLINST`	If set, flushes the instruction cache.
Bit 7	`C_INVINNST`	If set, invalidates the instruction cache.
Bit 8	`C_ADDR`	If set, flags a target address for flush operation.
Bits 9-14		Reserved for future use by Microware.
Bit 15	`C_GETCCTL`	If set, returns a pointer to the cache control routine and cache static global data.
Bit 16	`C_STODATA`	If set, stores the data cache (if supported by hardware).
Bits 17-31		Reserved for future use by Microware.



Note

All other bits are reserved. If any reserved bit is set, an `EOS_PARAM` error is returned.

By OS-9 convention, the terms flush, invalidate, and store are defined as follows:

Flush	Operation by which a cache line is written to main memory and marked as invalid.
Invalidate	Operation by which a cache line is marked as invalid. No write of cache line to main memory is performed.
Store	Operation by which a cache line is written to main memory. The cache line is not marked as invalid.

These specific operations may or may not be supported by a particular hardware configuration. The Flush operation is considered the base operation and will be supported by most processors.

`cctl` is a pointer to the location where `_os_scache()` stores the cache routine's address.

`cctl_data` is a pointer to the location where `_os_scache()` stores the cache routine's static storage address.

`addr` is a pointer that specifies the address for the flush operation. This parameter is used when the `C_ADDR` bit of `control` is set. Set `addr` to 0 when `C_ADDR` is not set.

`size` indicates the size of the target memory area to be flushed. This parameter is used when the `C_ADDR` bit of `control` is set. Set `size` to 0 when `C_ADDR` is not set.

Any program that builds or changes executable code in memory should flush the instruction cache with `_os_scache()` before executing new code. This is necessary because the hardware instruction cache is not updated by data (`write`) accesses and may contain the unchanged

instruction(s). For example, if a subroutine builds a system call on its stack, you must use the `_os_scache()` system call to flush the instruction cache before executing the temporary instructions.

If the `C_GETCCTL` bit of `control` is set, `_os_scache()` returns a pointer to the cache control routine in the cache extension module and a pointer to that routine's static global data. This enables drivers and file managers to call the cache routine directly rather than making a possibly time-consuming `_os_scache()` request. The prototype for this cache control routine is as follows:

```
error_code (*cache_cctl) (u_int32 control,
                          void *addr, u_int32 size);
```

The control, addr, and size parameters are as defined earlier. The only exception is the `C_GETCCTL` bit in the `control` parameter. This obviously has no meaning in this context.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`I_SETSTAT, SS_CACHE`	***OS-9 Technical Manual***
`F_CCTL (System State)`	***OS-9 Technical Manual***

Syntax

```
#include <memory.h>
#include <types.h>
error_code _os9_schbit(
     u_int16     *bit_number,
     u_int16     *count,
     void        *address,
     void        *bitmap_end);
```

Description

`_os9_schbit()` searches the specified buffer which represents a bit map for a free block (cleared bits) of the required length, starting at the beginning bit number. `_os9_schbit()` returns the offset of the first block found of the specified length.

The beginning bit number to search is passed at the location pointed to by `bit_number`. `_os9_schbit()` also stores the beginning bit number found at the location pointed to by `bit_number`.

The number of bits needed is passed at the location pointed to by count. `_os9_schbit()` also stores the number of bits found at the location pointed to by count.

`address` is a pointer to the base address of the bitmap.

`bitmap_end` is a pointer to the end of the bitmap plus one.

If `_os9_schbit()` returns SUCCESS, you should check `count` to make sure that you received the requested number of bits. If `count` is less than the requested amount, `_os9_schbit()` returned the largest number of bits available, starting at `bit_number`.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

_os9_allbit()
_os9_delbit()

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_seek(
     path_id     path,
     u_int32     position);
```

Description

`_os_seek()` repositions the path's file pointer. The file pointer is the 32-bit address of the next byte in the file to read or write. A `seek` may be performed to any value, even if the file is not large enough. Subsequent `write` requests automatically expand the file to the required size, if possible. `read` requests return an end-of-file condition.

`path` specifies the path number.

`position` specifies the new position in the file.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`I$Seek`	***OS-9 for 68K Technical Manual***
`I_SEEK`	***OS-9 Technical Manual***

_os_sema_init()

Initialize Semaphore Data Structure for Use

Syntax

```
#include <semaphore.h>
error_code _os_sema_init(semaphore *sema);
```

Description

OS-9 for 68K: `_os_sema_init()` is valid in version 3.0 and greater.

`_os_sema_init()` initializes a semaphore data structure for subsequent use. The application should call `_os_sema_init()` before the first use of the semaphore. The semaphore may be part of a data module or it may be part of any data structure within the application.

`sema` is a pointer to the semaphore data structure to initialize.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_sema_p()`
`_os_sema_term()`
`_os_sema_v()`

_os_sema_p()

Reserve Semaphore (Acquire Exclusive Access)

Syntax

```
#include <semaphore.h>
error_code _os_sema_p(semaphore *sema);
```

Description

OS-9 for 68K: `_os_sema_p()` is valid in version 3.0 and greater.

`_os_sema_p()` attempts to reserve the semaphore. If the semaphore is busy, the caller is suspended until the current holder releases the semaphore.

`sema` is a pointer to the semaphore to reserve.

`_os_sema_p()` returns an error code if a signal arrives while a process is waiting for a semaphore (i.e., it signals that the process is resuming execution--because of the incoming signal--without obtaining the semaphore).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_sema_init()`
`_os_sema_term()`
`_os_sema_v()`

_os_sema_term()

Terminate Use of Semaphore Data Structure

Syntax

```
#include <semaphore.h>
error_code _os_sema_term(semaphore *sema);
```

Description

OS-9 for 68K: `_os_sema_term()` is valid in version 3.0 and greater.

`_os_sema_term()` terminates the use of the semaphore. The application should call `_os_sema_term()` before terminating the application.

`sema` is a pointer to the semaphore data structure to terminate.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_sema_init()`
`_os_sema_p()`
`_os_sema_v()`

Syntax

```
#include <semaphore.h>
error_code _os_sema_v(semaphore *sema);
```

Description

OS-9 for 68K: `_os_sema_v()` is valid in version 3.0 and greater.

`_os_sema_v()` releases the reservation of a semaphore. If a process is waiting on the semaphore, it is activated, allowing it to attempt to acquire the semaphore.

`sema` is a pointer to the semaphore to release.

The `_os_sema_v()` call releases the semaphore, but does not give it to the next process waiting for it. Instead, if the process that issued the `_os_sema_v()` call issues a `_os_sema_p()` call before its slice is up, then it re-obtains the semaphore (i.e., the waiting process if running at the same priority was unable to obtain it).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_sema_init()`
`_os_sema_p()`
`_os_sema_term()`

_os_send()

Send Signal to Another Process

Syntax

```
#include <signal.h>
#include <types.h>
error_code _os_send(
     process_id   proc_id,
     signal_code  signal);
```

Description

`_os_send()` sends a signal to the specified process. A process may send the same signal to all processes of the same group/user ID by passing zero as the receiving process' ID number. `proc_id` is the process ID number for the intended receiver. `signal` specifies the signal code to send.

Only the lower 16 bits of the signal value are used. The upper 16 bits are ignored.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_wait()`
`_os_intercept()`
`_os_sleep()`

F$Send	***OS-9 for 68K Technical Manual***
F_SEND	***OS-9 Technical Manual***

_os_setcrc()

Generate Valid CRC in Module

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_setcrc(mh_com *mod_head);
```

Description

`_os_setcrc()` updates the header parity and CRC of a module in memory. The module must have the correct size and sync bytes; other parts of the module are not checked. `mod_head` is a pointer to the module. The module image must start on a longword address or an exception may occur.

WARNING

Although `_os_setcrc()` and `_setcrc()` functions update the `header` and `crc` of a module in memory, the copy of the module header in the module directory is not affected. Therefore, you have an invalid module directory entry for the memory module with the new CRC; and an error 236 results if links to the module are attempted.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_crc()`

_os_setime()

Set System Time

Syntax

```
#include <time.h>
#include <types.h>
error_code _os_setime(u_int32 time);
```

Description

`_os_setime()` sets the current system time and starts the system real-time clock to produce time slice interrupts. Only super users may set the time.

`time` is stored as the number of seconds since a specific date in Greenwich Mean Time. `time` is the desired current time in Greenwich Mean Time.

For More Information

To determine the specified date, reference the applicable manual, ***OS-9 for 68K Technical Manual*** or ***OS-9 Technical Manual.***

On OS-9 for 68K, the binding for this call performs conversions which allow compatibility with OS-9. If speed is a consideration, refer to `_os9_setime()`.

The time is not validated in any way. If time is zero on systems with a battery-backed clock, the actual time is read and set from the real-time clock.

In order to get the system time/date from the real time clock or the battery-backed clock, the month field in the date parameter must be `0`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`gmtime()`	
`_os_link()`	
`_os_getime()`	
`_os9_setime()`	
`F$STime`	***OS-9 for 68K Technical Manual***
`F_STIME`	***OS-9 Technical Manual***

_os9_setime()

Set System Date/Time

Syntax

```
#include <time.h>
#include <types.h>
error_code _os9_setime(
     u_int32     time,
     u_int32     date);
```

Description

`_os9_setime()` sets the current system date/time and starts the system real-time clock to produce time slice interrupts. Only super users may set the time.

`time` is in the format `00hhmmss`.

`date` is in the format `yyyymmdd`.

Each value is one byte in length, except for the year which is 2 bytes long. Values must be in hexadecimal format.

The date and time are not checked for validity. On systems with a battery-backed clock, it is usually only necessary to supply the year to the `_os9_setime()` call. The actual date and time are read from the real-time clock.

In order to get the system time/date from the real-time clock or the battery-backed clock, the month field in the date parameter must be `0`.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

_os_link()
_os9_getime()
F$STime *OS-9 for 68K Technical Manual*

_os_setpr()

Set Process Priority

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_setpr(
     process_id   proc_id,
     u_int32      priority);
```

Description

`_os_setpr()` changes the process priority to the value specified by priority. A super user (group ID zero) may change any process' priority. A non-super user can only change the priorities of processes with the same user ID.

`proc_id` is the process ID.

`priority` specifies the new priority. `65535` is the highest priority; `0` is the lowest.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$SPrior`	***OS-9 for 68K Technical Manual***
`F_SPRIOR`	***OS-9 Technical Manual***

_os_setstat()

Set File/Device Status

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_setstat(
     path_id     path,
     u_int32     code,
     void        *pb);
```

Description

`_os_setstat()` is a wildcard call used to handle individual device parameters that are not uniform on all devices or are highly hardware dependent.

`path` is the path number. `code` is the `SetStat` code. Only the lower 16 bits of code are used. `pb` is a pointer to the parameter block.

The exact operation of this call depends on the device driver and file manager associated with the path. The mnemonics for the status codes are located in the `sg_codes.h` header file. This manual describes the status codes currently defined by Microware and the functions that they perform. These calls have an `_os_ss_` prefix.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <svctbl.h>
#include <types.h>
error_code _os_setsvc(
     u_int32     count,
     u_int32     state_flag,
     void        *init_tbl,
     void        *params);
```

Description

`_os_setsvc()` adds or replaces a system service in OS-9's user and privileged system service request tables.

`count` is a count of the entries in the initialization table.

`state_flag` specifies whether user or system state tables, or both, are updated.

If `state_flag` is `USER_STATE (1)`, only the user table is updated.

If `state_flag` is `SYSTEM_STATE (2)`, only the system table is updated.

If `state_flag` is `USER_STATE` OR'ed with `SYSTEM_STATE (3)`, both tables are updated.

`init_tbl` is a pointer to the initialization table. An example initialization table might look like this:

```
error_code printmsg(), service();

svctbl syscalls[] =
{
   {F_PRINT, printmsg,  0, 1, 1},
   {F_SERVICE, service, 0 , 1, 1}
};
```

`params` may be a pointer to anything, but is intended to be a pointer to global static storage. When a system call is used, the `params` data pointer is used as the global static storage pointer for the function.

Note

Use `get_static()` from `cpu.l` to acquire your current static storage pointer.

Valid service request codes range from `0` to `255`.

The available service request codes are located in the `funcs.h` file.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <funcs.h>
#include <types.h>
error_code _os9_setsvc(
     void         *init_tbl,
     void         *params);
```

Description

`_os9_setsvc()` adds or replaces function requests in OS-9 for 68K's user and privileged system service request tables. To install a system-state-only system call, SYSTRAP is added to the function code number. Otherwise, both the system- and user-state dispatch tables are updated. SYSTRAP is defined in the `funcs.h` header file.

`init_tbl` is a pointer to the initialization table. The following is an example initialization table:

```
struct svc
{
 u_int16 func_code;   * service call function code number
u_int16 offset;         * difference between the address of
the
                       * next table entry and the address of
the
                         * routine.*/
} svctbl[] =


{
 {F_SERVICE,(u_int32)routine -(u_int32)&svctbl[1]},
            {F_SERVICE+SYSTRAP,
          (u_int32)sys_routine -(u_int32)&svctbl[2]};
  {-1,-1}
};
```

Valid service request codes range from `0` to `255`.

If the most significant bit of the function code word is set, only the system table is updated. Otherwise, both the system and user tables are updated.

`params` is intended to be a pointer to the global static storage. This allows you to associate a global data pointer with each installed system call. When the system call is used, the data pointer is automatically passed. Whatever `params` points to is passed to the system call; `params` may be a pointer to anything.

Note

The OS-9 for 68K service request codes are located in `/MWOS/OS9/SRC/DEFS`.

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_setsys()

Set System Global Variables

Syntax

```
#include <sysglob.h>
#include <types.h>
error_code _os_setsys(
     u_int32     offset,
     u_int32     size,
     glob_buff   sysvar);
```

Description

`_os_setsys()` changes a system global variable. These variables have a d_ prefix and are located in the system library `sysglob.h`. Consult the files in the DEFS directory for a description of the system global variables. Only super users may change system variables. Super users must be extremely careful when changing system global variables. `offset` is the offset to the system global variables. `size` specifies the size of the target variable. `sysvar` is the new value for the system global variable.

For More Information

Refer to your operating system technical manual for the system global variables that may be changed.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_setuid()

Set User ID Number

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_setuid(owner_id user_id);
```

Description

`_os_setuid()` changes the current user ID to `user_id`.

`user_id` is the desired group/user ID number.

The following restrictions apply to `_os_setuid()`:

- Users with group ID zero may change their IDs to anything without restriction.
- A primary module owned by a group zero user may change its ID to anything without restriction.
- Any primary module may change its user ID to match the module's owner.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_sgetstat()

GetStat Call Using System Path Number

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_sgetstat(
     path_id      spath,
     u_int32      code,
     void         *pb);
```

Description

`_os_sgetstat()` makes a specific `GetStat` or `SetStat` call.

`spath` is the system path number.

`code` is the `GetStat` code.

`pb` is a pointer to the parameter block corresponding to the function being performed. If the function does not have a parameter block (for example, `_os_ss_reset()`), `pb` should be `NULL`.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_sgs_devnm()

Return Device Name

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_sgs_devnm(
     path_id     path,
     char        *namebuf);
```

Description

`_os_sgs_devnm()` returns the name of the device associated with the specified system path number.

`path` is the system path number.

`namebuf` is a pointer to the location where `_os_sgs_devnm()` stores the device name.

Attributes

Operating System:	OS-9K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_siglngj()

Set Signal Mast Value and Return on Specified Stack Image

Syntax

```
#include <signal.h>
#include <types.h>
error_code _os_siglngj(
     void         *usp,
     u_int16      siglvl);
```

Description

F_SIGLNGJ allows processes to perform longjump() operations from their signal intercept routines and unmask signals in one operation.

usp is a pointer to the new process stack image.

siglvl is the new signal level value.

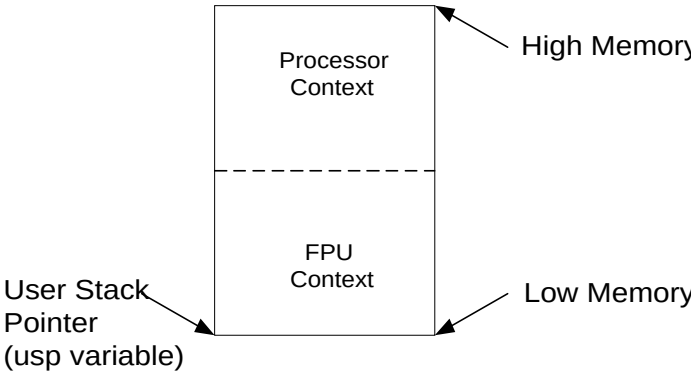
This call is usually used by nested intercept routines to resume execution in the process at a different location from where the process was interrupted by the original signal. When this call is made, the operating system performs the following functions:

- Validates and copies the target process stack image from the memory buffer pointed to by the usp variable to the process' system-state stack
- Sets the process' signal mask to the value specified in the siglvl variable
- Returns to the process restoring the context copied from the user state process stack image

The operating system takes appropriate precautions to verify that the memory location pointed to by the usp variable is accessible to the process and to ensure that the process does not attempt to make a state change.

The stack image pointed to by the `usp` variable must have the format shown in **Figure 2-9**.

Figure 2-9 _os_siglngj() Required Stack Image



The specific format of the processor context is defined by the `longstk` structure definition found in the `reg<CPU Family>.h` file for the associated processor. The format of the floating-point context varies depending on whether the target system has a hardware floating-point unit versus floating-point emulation software.

For floating-point hardware, the stack image is the same as that defined by the `fregs` structure definition found in the associated `reg<CPU Family>.h` header file.

For floating-point emulation, the floating-point context differs from the hardware implementation context in that it may contain additional context information specific to the FPU module performing the emulation. The definition for the floating-point context as used by the FPU module is the `fpu_context` structure which is also defined in the associated `reg<CPU Family>.h` header file for the target processor.

If a particular application needs to access the contents of the process context, it may use the size of these structures for indexing. Alternatively, the application can determine the size of the FPU context at runtime by accessing the kernel globals field, `d_fpusize`, which contains the size of the FPU context.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

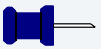
`os_lib.l`

See Also

`_os_sigmask()`
`_os_send()`

Syntax

```
#include <signal.h>
#include <types.h>
error_code _os_sigmask(int32 mode);
```



Note

The only function that `_os_sigmask` provides in system-state processes is to wake up that process. It does not interrupt code that is currently executing.

Description

`_os_sigmask()` enables signals to reach the calling process or disables signals from reaching the calling process. If a signal is sent to a process whose mask is disabled, the signal is queued until the process mask becomes enabled.

`mode` is the process signal level.

Table 2-22 _os_sigmask() Parameter Mode Values

If Mode Is	The Mask Level Is
`0`	Cleared
`1`	Set or incremented
`-1`	Decremented

Note

Signals are analogous to hardware interrupts and should be masked sparingly.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_sleep()`
`_os_wait()`

Example

To ensure that an entire structure can be cleared without the reception of a signal one might use:

```
_os_sigmask(1);
memset(sptr, 0, sizeof(*sptr));
_os_sigmask(-1);
```

The state of the signal mask is preserved using this method.

To avoid a race condition:

```
_os_sigmask(1);
sig_received = FALSE;
sleep(0);           * sleep does an implied
                  * _os_sigmask(0);
```

_os_sigreset()

Reset Signal Intercept Context Stack

Syntax

```
#include <signal.h>
error_code _os_sigreset(void);
```

Description

`_os_sigreset()` resets the context stack for a signal intercept. Normally, the operating system keeps the state of the process on the system stack while a signal intercept routine executes. If signals are unmasked during the intercept routine and another signal occurs before the intercept routine finishes, another state is pushed on the system stack.

Similarly, if the intercept routine does a `longjmps()`and then unmasks signals, more states are placed on the system stack by subsequent signals.

To prevent the system stack from overflowing, whenever a program uses `longjmp()` calls out of an intercept routine or unmasks signals in an interrupt service routine with the intent of never using `_os_rte()` to return, it should use `_os_sigreset()`:

```
if(setjmp(x) != 0) {
    _os_sigreset();
    _os_sigmask(-1);
}
```

If the code actually expects to return through the nested intercept routines with multiple `_os_rte()` calls, the states must be left where they are and `_os_sigreset()` should not be called.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

_os_rte()

_os_sigrs()

Resize Current Process' Signal Queue Block

Syntax

```
#include <signal.h>
#include <types.h>
error_code _os_sigrs(u_int32 signals);
```

Description

`_os_sigrs()` allows a process to change the maximum number of signals that are queued on its behalf.

`signals` is the new maximum number of signals.

`_os_sigrs()` can be used to increase or decrease the number of signals queued. An error is returned (EOS_PARAM) if a request is made to reduce the number of queued signals while there are signals pending. The initial default for the system is specified in the system `Init` module.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_sleep()

Put Calling Process to Sleep

Syntax

```
#include <signal.h>
#include <types.h>
error_code _os_sleep(
     u_int32      *ticks,
     signal_code  *signal);
```

Description

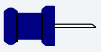
`_os_sleep()` deactivates the calling process until the requested number of ticks have elapsed.

The length of time to sleep in ticks/second is passed at the location pointed to by `ticks`.

- If `ticks` is zero, the process sleeps indefinitely.
- If `ticks` is one, the process gives up a time slice but does not necessarily sleep for one tick.
- `_os_sleep()` stores the number of ticks left to sleep when awakened prematurely at the location pointed to by `ticks`. If `_os_sleep()` completes without interruption, `ticks` contains the value `0`.

`time` may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second.

`signal` is a pointer to the location where `_os_sleep()` stores the last signal the process received.



Note

`_os_sleep()` cannot be used to time more accurately than ± 1 tick because it is not known when the `_os_sleep()` request was made during the current tick.

Note

The system clock must be running to perform a timed sleep. The system clock is not required to perform an indefinite sleep or to give up a time slice.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_send()`
`_os_wait()`

_os9_sleep()

Put Calling Process to Sleep

Syntax

```
#include <signal.h>
#include <types.h>
error_code _os9_sleep(u_int32 *ticks);
```

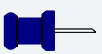
Description

`_os9_sleep()` deactivates the calling process until the requested number of ticks have elapsed.

The requested number of ticks is passed at the location pointed to by `ticks`.

- If `ticks` is zero, the process sleeps indefinitely.
- If `ticks` is one, the process gives up a time slice but does not necessarily sleep for one tick.
- `_os9_sleep()` stores the number of ticks left to sleep when awakened prematurely at the location pointed to by ticks. If `_os9_sleep()` completes without interruption, `ticks` contains the value `0`.

`time` may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second.



Note

`_os9_sleep()` cannot be used to time more accurately than ±1 tick because it is not known when the `_os9_sleep()` request was made during the current tick.

Note

The system clock must be running to perform a timed sleep. The system clock is not required to perform an indefinite sleep or to give up a time slice.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_send()`
`_os_wait()`

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_slink(
     u_int32      sub_num,
     char         *mod_name,
     void         **lib_exec,
     void         **mem_ptr,
     mh_com       **mod_head);
```

Description

`_os_slink()` attempts to link or load the named module. It returns a pointer to the execution entry point and a pointer to the library's static data area for subsequent calls to the subroutine. The calling program must store and maintain the subroutine's entry point and data pointer. The calling program must also set the subroutine library's data pointer and dispatch to the correct subroutine.

`sub_num` is the subroutine number. Only the lower 16 bits of `sub_num` are used.

`mod_name` is a pointer to the name of the subroutine module.

`lib_exec` is a pointer to the location where `_os_slink()` stores the pointer to the subroutine entry point.

`mem_ptr` is a pointer to the location where `_os_slink()` stores the pointer to the subroutine static memory.

`mod_head` is a pointer to the location where `_os_slink()` stores the pointer to the module header.

You can remove a subroutine by passing a null pointer for the name of the module and specifying the subroutine number.

A process can link to a maximum of 16 subroutine libraries, numbered from `0` to `15`.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_tlink()`

_os_slinkm()

Install User Subroutine Library Module by Module Pointer

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_slinkm(
     u_int32      sub_num,
     mh_com       *mod_head,
     void         **lib_exec,
     void         **mem_ptr);
```

Description

`_os_slinkm()` is passed a pointer to the subroutine library module to install. It returns a pointer to the execution entry point and a pointer to the library's static data area for subsequent calls to the subroutine. The calling program must store and maintain the subroutine's entry point and data pointer. The calling program must also set the subroutine library's data pointer and dispatch to the correct subroutine.

`sub_num` is the subroutine number. Only the lower 16 bits of `sub_num` are used.

`mod_head` is a pointer to the module header.

`lib_exec` is a pointer to the location where `_os_slinkm()` stores the pointer to the subroutine library entry point.

`mem_ptr` is a pointer to the location where `_os_slinkm()` stores the pointer to the subroutine library static memory.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

_os_slink()
_os_tlink()

_os_srqmem()

System Memory Request

Syntax

```
#include <memory.h>
#include <types.h>
error_code _os_srqmem(
     u_int32      *size,
     void         **mem_ptr,
     u_int32      color);
```

Description

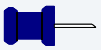
`_os_srqmem()` allocates a block of a specific memory type.

The byte count of the requested memory is passed at the location pointed to by `size`. `_os_srqmem()` also stores the actual number of bytes allocated at the location pointed to by size. If size is `0xFFFFFFFF`, the largest block of free memory of the specified type is allocated to the calling process.

`mem_ptr` is a pointer to the location where `_os_srqmem()` stores the pointer to the allocated memory block.

`color` specifies the memory type. Only the lower 16 bits of `color` are used.

- If `color` is non-zero, the search is restricted to memory areas of that color. The area with the highest priority is searched first.
- If `color` is zero, the search is based only on priority.



Note

The byte count of allocated memory and the pointer to the allocated block must be saved if the memory is ever to be returned to the system.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os9_srqmem()`
`_os_srtmem()`
`_os_mem()`

`F$SRqCMem`	***OS-9 for 68K Technical Manual***
`F_SRQMEM`	***OS-9 Technical Manual***

_os9_srqmem()

System Memory Request

Syntax

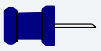
```
#include <memory.h>
#include <types.h>
error_code _os9_srqmem(
     u_int32      *size,
     void         **mem_ptr);
```

Description

`_os9_srqmem()` allocates a memory block from the top of available RAM.

The size of the memory block is passed at the location pointed to by `size`. `_os9_srqmem()` also stores the actual number of bytes allocated at the location pointed to by size. If size is `0xFFFFFFFF`, the largest block of free memory is allocated to the calling process.

`mem_ptr` is a pointer to the location where `_os9_srqmem()` stores the pointer to the allocated memory block.



Note

The byte count of allocated memory and the pointer to the allocated block must be saved if the memory is ever to be returned to the system.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

_os_srtmem()
_os_mem()
_os_srqmem()
F$SRqMem *OS-9 for 68K Technical Manual*

_os_srtmem()

Return System Memory

Syntax

```
#include <memory.h>
#include <types.h>
error_code _os_srtmem(
     u_int32      size,
     void         *mem_ptr);
```

Description

`_os_srtmem()` deallocates memory when it is no longer needed.

`size` specifies the byte count of the returned memory.

`mem_ptr` is a pointer to the memory block to return.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_srqmem()`
`_os_mem()`

_os_ss_attr()

Set File Attributes

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_attr(
     path_id     path,
     u_int32     attr);
```

Description

`_os_ss_attr()` changes a file's attributes to the new value, if possible. You cannot set the directory bit of a non-directory file or clear the directory bit of a non-empty directory.

`path` specifies the path number of the file.

`attr` specifies the new values for the file attributes.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_break()

Send Break Conditional to Serial Line

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_break(path_id path);
```

Description

`_os_ss_break()` sends a break conditional to the serial line.

`path` specifies the path number.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`I_SETSTAT, SS_BREAK` ***OS-9 Technical Manual***

_os_ss_cache()

Enable/Disable RBF Caching

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_ss_cache(
     path_id     path,
     u_int32     enblflag,
     u_int32     drvcsize);
```

Description

`_os_ss_cache()` enables and disables RBF caching on an RBF device.

`path` specifies the path number.

`enblflag` is the cache enable/disable flag.

- If `enblflag` is zero, caching is disabled.
- If `enblflag` is non-zero, caching is enabled.

`drvcsize` is the memory size for the cache.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os9_ss_close()

Notify Driver that Path Has Been Closed

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os9_ss_close(path_id path);
```

Description

`_os9_ss_close()` notifies a driver that a path has been closed.

`path` specifies the path number that was closed.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_dcoff()

Set Signal to Be Sent when DCDL Goes False

Syntax

```
#include <scf.h>
#include <types.h>
error_code _os_ss_dcoff(
     path_id     path,
     signal_code  signal);
```

Description

`_os_ss_dcoff()` sets the signal code to be sent when the Data Carrier Detect line becomes false.

`path` specifies the path number.

`signal` is the signal code to send.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_ss_dcon()`

_os_ss_dcon()

Set Signal to Be Sent when DCDL Goes True

Syntax

```
#include <scf.h>
#include <types.h>
error_code _os_ss_dcon(
     path_id      path,
     signal_code  signal);
```

Description

`_os_ss_dcon()` sets the signal code to be sent when the Data Carrier Detect line becomes true.

`path` specifies the path number.

`signal` is the signal code to send.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_ss_dcoff()`

_os_ss_dopt()

Set Device Path Options

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_dopt(
     path_id     path,
     u_int32     size,
     void        *dopts);
```

Description

`_os_ss_dopt()` sets the initial (default) device path options (`dopts`). These options initialize new paths to the device.

`path` specifies the path number.

`size` specifies the size of the options area.

`dopts` is a pointer to the default options for the device.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_dsrts()

Disable RTS Line

Syntax

```
#include <scf.h>
#include <types.h>
error_code _os_ss_dsrts(path_id path);
```

Description

`_os_ss_dsrts()` disables the RTS line on serial devices.

`path` specifies the path number.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`I$SetStt`	***OS-9 for 68K Technical Manual***
`I_SETSTAT, SS_DSRTS`	***OS-9 Technical Manual***

_os_ss_enrts()

Enable RTS Line

Syntax

```
#include <scf.h>
#include <types.h>
error_code _os_ss_enrts(path_id path);
```

Description

`_os_ss_enrts()` enables the RTS line on serial devices.

`path` specifies the path number.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`I$SetStt`	***OS-9 for 68K Technical Manual***
`I_SETSTAT, SS_ENRTS`	***OS-9 Technical Manual***

_os_ss_erase()

Erase Tape

Syntax

```
#include <sbf.h>
#include <types.h>
error_code _os_ss_erase(
     path_id      path,
     u_int32      blocks);
```

Description

`_os_ss_erase()` erases a portion of the tape.

`path` specifies the tape drive's path number.

`blocks` specifies the number of blocks to erase.

- If `blocks` is `0xffffffff`, SBF erases until the end-of-tape is reached.
- If `blocks` is positive, SBF erases the amount of tape equivalent to that number of blocks.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_ss_fd(
     path_id      path,
     fd_stats     *fdinfo);
```

Description

`_os_ss_fd()` changes the file descriptor sector data. The path must be open for write. To change the file descriptor sector data, you must first `read` the sector, `make` the changes, and then `write` the sector.

`path` specifies the path number.

`fdinfo` is a pointer to the file descriptor's buffer.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_fillbuff()

Fill Path Buffer with Data

Syntax

```
#include <scf.h>
#include <types.h>
error_code _os_ss_fillbuff(
     path_id      path,
     char         *buffer,
     u_int32      size);
```

Description

`_os_ss_fillbuff()` fills the input path buffer with the data in buffer.

`path` specifies the path number.

`buffer` is a pointer to the location of your buffer.

`size` specifies the size of your buffer.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_flushmap()

Flush Cached Bit Map Information for RBF Device

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_ss_flushmap(
     path_id      path);
```

Description

`_os_ss_flushmap()` flushes the cached bit map information for an RBF device. This is normally only performed after the bit `map` on the disk is changed by a utility such as format.

`path` is the path number for any file on the device or the entire device (for example, a path open to `/h0@`).

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_hdlink()

Make Hard Link to Existing File

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_ss_hdlink(
     path_id      path,
     char         **link_path);
```

Description

`_os_ss_hdlink()` creates a new directory entry. This directory entry points to the file descriptor block of an open file. `_os_ss_hdlink()` updates the pathlist pointer.

`path` is the path number of an existing file.

The new name for the directory entry is passed at the location pointed to by `link_path`. `_os_ss_hdlink()` also stores the updated pathlist pointer at the location pointed to by `link_path.`

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_link_adj()(
     path_id      path,
     u_in32       fd_addr,
     int32        link_adj)
```

Description

This I_SETSTAT call sets the link count of a file. The link count of a file indicates the number of hard links in the file system that point to a particular file. This call can be used to correct the link count of files under corrupt file system conditions. File link counts can only be adjusted by super-user processes.

Attributes

Operating System:	OS-9
State:	User; System; I/O; IRQ
Threads:	Safe
Re-entrant:	Yes

Library

oslib.l

See Also

_os_ss_hdlink()
_os_setstat()

_os_ss_lock()

Lock Out Record

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_ss_lock(
     path_id      path,
     u_int32      size);
```

Description

`_os_ss_lock()` locks out a section of the file from the current file pointer position up to the specified number of bytes.

`path` is the file's path number.

`size` is the size of the section to lockout.

- If `size` is zero, all locks are removed (record lock, EOF lock, and file lock).
- If `size` is `$ffffffff`, the entire file is locked out regardless of the file pointer's location. This is a special type of file lock that remains in effect until released by an `_os_ss_lock()` with `size` set to zero, a `read` or `write` of zero bytes, or the file is closed.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_luopt()

Write Logical Unit Options

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_luopt(
     path_id     path,
     u_int32     size,
     void        *luopts);
```

Description

`_os_ss_luopt()` writes the logical unit options for a path to a buffer.

`path` is the path number.

`size` specifies the buffer size of the logical unit options area.

`luopts` is a pointer to the logical unit options.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os9_ss_open()

Notify Driver that Path Has Been Opened

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os9_ss_open(path_id path);
```

Description

`_os9_ss_open()` is an internal call for drivers.

`path` specifies the path number.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_popt(
     path_id      path,
     u_int32      size,
     void         *user_opts);
```

Description

`_os_ss_popt()` writes the option section of the path descriptor from the options buffer pointed to by `user_opts`.

`path` is the path number.

`size` specifies the buffer size.

`user_opts` is a pointer to the options buffer.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_gs_popt()`	
`I$SetStt`	***OS-9 for 68K Technical Manual***
`I_SETSTAT`	***OS-9 Technical Manual***

_os_ss_relea()

Release Device

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_relea(path_id path);
```

Description

`_os_ss_relea()` releases the device from any `_os_ss_sendsig()`, `_os_ss_dcon()`, or `_os_ss_dcoff()` request made by the calling process.

`path` is the path number of the device to release.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_ss_sendsig()`
`_os_ss_dcoff()`
`_os_ss_dcon()`

_os_ss_rename()

Rename File

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_rename(
     path_id     path,
     char        *newname);
```

Description

`_os_ss_rename()` changes the file name in the file's associated directory entry to `newname`.

`path` is the file's path number.

`newname` is a pointer to the file's new name.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_reset()

Restore Head to Track Zero

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_reset(
     path_id      path);
```

Description

For RBF, `_os_ss_reset()` directs the disk head to track zero. It is used for formatting and error recovery. For SBF, `_os_ss_reset()` rewinds the tape.

`path` is the path number of the device.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_reten()

Retension Pass on Tape Drive

Syntax

```
#include <sbf.h>
#include <types.h>
error_code _os_ss_reten(path_id path);
```

Description

`_os_ss_reten()` performs a retention pass on the tape drive.

`path` is the path number of the tape drive.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_rfm()

Skip Tape Marks

Syntax

```
#include <sbf.h>
#include <types.h>
error_code _os_ss_rfm(
     path_id      path,
     int32        marks);
```

Description

`_os_ss_rfm()` skips the number of tape marks specified in marks.

`path` is the tape drive's path number.

`marks` specifies the number of tape marks to skip. If marks is negative, the tape is rewound the specified number of marks.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_sendsig()

Send Signal on Data Ready

Syntax

```
#include <sg_codes.h>
#include <types.h
error_code _os_ss_sendsig(
     path_id     path,
     signal_code  signal);
```

Description

`_os_ss_sendsig()` sets up a signal to send to a process when an interactive device or pipe has data ready. `_os_ss_sendsig()` must be reset each time the signal is sent.

`path` specifies the path number of the process receiving the signal.

`signal` specifies the signal to send.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_ss_relea()`

_os_ss_size()

Set File Size

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_size(
     path_id      path,
     u_int32      size);
```

Description

`_os_ss_size()` sets the size of the file associated with `path` to the specified size.

`path` is the file's path number.

`size` is the size of the file in bytes.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <sbf.h>
#include <types.h>
error_code _os_ss_skip(
     path_id     path,
     int32       blocks);
```

Description

`_os_ss_skip()` skips the specified number of blocks.

`path` is the path number of the tape drive.

`blocks` specifies the number of blocks to skip. If `blocks` is negative, the tape is rewound the specified number of blocks.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_skipend()

Skip to End of Tape

Syntax

```
#include <sbf.h>
#include <types.h
error_code _os_ss_skipend(path_id path);
```

Description

`_os_ss_skipend()` skips the tape to the end of data. This enables you to append data to tapes on cartridge-type tape drives.

`path` is the path number of the tape drive.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_ticks()

Wait Specified Number of Ticks for Record Release

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_ss_ticks(
     path_id      path,
     u_int32      delay);
```

Description

`_os_ss_ticks()` may be used to cause an error (`EOS_LOCK`) to return to the user program if a file lock conflict still exists after the specified number of ticks have elapsed.

`path` is the path number.

`delay` specifies the delay interval.

- A delay of zero (RBF's default) causes sleep until the record is released.
- A delay of one indicates that if the record is not released immediately, an error is returned. If the high order bit is set, the lower 31 bits are converted from 256ths of a second to ticks before sleeping.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_wfm()

Write Tape Marks

Syntax

```
#include <sbf.h>
#include <types.h>
error_code _os_ss_wfm(
     path_id      path,
     u_int32      marks);
```

Description

`_os_ss_wfm()` writes the specified number of tape marks.

`path` specifies the tape drive's path number.

`marks` specifies the number of tape marks to write.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_ss_wtrack(
     path_id     path,
     void        *trkbuf,
     void        *ilvtbl,
     u_int32     track,
     u_int32     head,
     u_int32     interleave);
```

Description

`_os_ss_wtrack()` causes a format track operation (used with most floppy disks) to occur. For hard or floppy disks with a “format entire disk” command, this formats the entire media only when the track number and side number are both zero.

`path` is the path number.

`trkbuf` is a pointer to the track buffer.

`ilvtbl` is a pointer to the interleave table. The interleave table contains byte entries of LBNs ordered to match the requested interleave offset.

`track` is the track number.

`head` is the side number.

`interleave` is the interleave value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

I$SetStt	***OS-9 for 68K Technical Manual***
I_SETSTAT	***OS-9 Technical Manual***

Syntax

```
#include <settrap.h>
#include <types.h>
#include <regs.h>
error_code _os_strap(
     u_int32      *excpt_stack,
     strap        *init_tbl);
```

Description

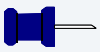
`_os_strap()` enables user programs to catch “process local” program exceptions such as illegal instructions and divide-by-zeroes.

`excpt_stack` is a pointer to the stack to use if an exception occurs. If `excpt_stack` is zero, `_os_strap()` uses the current stack.

`init_tbl` is a pointer to the service request initialization table.

If a user routine is not provided and one of these exceptions occurs, the program is aborted. An initialization table for a 68K platform might appear as follows:

```
strap errtab[] = {
    {T_BUSERR, errtrap},
    {T_ADDERR, errtrap},
    {-1, NULL}
};
```



Note

The previous example is for the 68K platform only. The `T_BUSERR` and `T_ADDERR` are defined only for that platform, not for the PPC or X86 platforms.

When a user’s exception routine is executed, it is passed the following:

```
void errtrap(vector_errno, badpc, badaddr)
  u_int32 vector_errno,  * error number of the vector

       badpc,            * PC where exception occurred
        badaddr;          * address where exception
                          * occurred
```

You can disable an error exception handler by calling `_os_strap()` with an initialization table that specifies zero as the offset to the routine(s) to remove.

Note

On OS-9 for 68K, the binding for this call performs conversions which allow compatibility with OS-9. If speed is a consideration, see `_os9_strap()`.

Note

Beware of exceptions in exception handling routines. They are usually not re-entrant.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$STrap`	***OS-9 for 68K Technical Manual***
`F_STRAP`	***OS-9 Technical Manual***

Syntax

```
#include <settrap.h>
#include <regs.h>
#include <types.h>
error_code _os9_strap(
     u_int32     *excpt_stack,
     os9strap    *init_tbl);
```

Description

`_os9_strap()` enables user programs to catch “process local” program exceptions such as illegal instructions and divide by zeroes.

`excpt_stack` is a pointer to the stack to use if an exception occurs. If `excpt_stack` is zero, `_os9_strap()` uses the current stack.

`init_tbl` is a pointer to the service request initialization table. The first field of the `init_tbl` should be the vector `OFFSET` of the exception vector on which to install the handler. The second field is the `OFFSET` from the next entry in `init_tbl` to the exception handling function for this vector. `init_tbl` may look like the following:

```
void ovferror();
void chkerror();
extern void os9_init_tbl();

os9strap *init_tbl = (os9strap *)os9_init_tbl;

_asm("os9strap_init_tbl: dc.w %0,
      ovferror-*-4", T_TRAPV << 2);
_asm(" dc.w %0, chkerror-*-4", T_CHK << 2);
_asm(" dc.w -1, -1");
```

Note

To access the contents of the initialization table, `os9strap_init_tbl` in this example, you must define `os9_init_tbl` as if it were a function.

For example, `os9strap *os9_init_tbl` would not generate PC-relative references, while `extern void os9_init_tbl()` forces the compiler to generate references into the code area.

If a user routine is not provided and one of these exceptions occur, the program is aborted.

You can disable an error exception handler by calling `_os9_strap()` with an initialization table that specifies zero as the offset to the routine(s) to remove.

Note

Beware of exceptions in exception handling routines. They are usually not re-entrant.

Attributes

Operating System:	OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$STrap` ***OS-9 for 68K Technical Manual***

_os_suspend()

Suspend Process

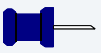
Syntax

```
#include <process.h>
#include <types.h>
error_code _os_suspend(process_id proc_id);
```

Description

`_os_suspend()` temporarily suspends a process. A super user (group ID zero) may suspend any process. A non-super user gets an `E_PERMIT` message returned.

`proc_id` identifies the target process.



Note

`_os_suspend()` is currently not supported on OS-9 for 68K.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_sysdbg()

Call System Debugger

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_sysdbg(
     void         *param1,
     void         *param2);
```

Description

`_os_sysdbg()` calls the system level debugger, if one exists.

`param1` and `param2` are parameters passed to the debugger. These are currently not used.

When the system level debugger is active, it runs in system state and effectively stops timesharing. When used in user state, `_os_sysdbg()` can only be called by users in group zero. When used in system state, anyone may use `_os_sysdbg()`. Never use this call when other users are on the system.

You must enable the system debugger before installing breakpoints or attempting to trace instructions. If no system debugger is available, the system is reset. The system debugger handles some of the exception vectors directly. This makes it impossible to use the user debugger when the system debugger is enabled.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <regs.h>
#include <types.h>
error_code _os_sysid(
     u_int32      *oem,
     u_int32      *serial,
     u_int32      *mpu_type,
     u_int32      *os_type,
     u_int32      *fpu_type,
     int32        *time_zone,
     u_int32      *kernel_ver,
     u_int32      *resv2,
     char         *sys_ident,
     char         *copyright,
     char         *author);
```

Description

`_os_sysid()` returns the following information about the system:

Table 2-23 _os_sysid() Return Information

Pointer	Information Stored by _os_sysid()
`oem`	OEM identification number
`serial`	Copy serial number
`mpu_type`	Processor identifier (such as 68020/68030/68040/80386)
`os_type`	Kernel (OS) MPU configuration

Table 2-23 _os_sysid() Return Information (continued)

Pointer	Information Stored by _os_sysid()
`fpu_type`	Floating-point processor identifier (such as 68881 and 80387)
`time_zone`	System time zone in minutes offset from Greenwich Mean Time (GMT)
`kernel_ver`	The kernel's version number for OS-9 for 68K
`resv2`	A reserved pointer
`sys_ident`	A pointer to an application buffer to copy the system identification message into. The buffer must be at least 80 bytes in length.
`copyright`	A pointer to an application buffer to copy the copyright message into. The buffer must be at least 80 bytes in length.
`author`	For OS-9: A pointer to an application buffer to copy the author message for OS-9. The buffer must be at least 80 bytes in length. For OS-9 for 68K: This parameter should be NULL.

Note

On OS-9 for 68K, `time_zone` is always returned as `-1`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_thexit

Exit a Thread

Syntax

```
#include <threads.h>
error_code _os_thexit(error_code status);
```

Description

`_os_thexit()` causes the calling thread to exit. If the calling program is not multi-threaded, the EOS_PERMIT error is returned.

`status` is the value to be returned to any thread waiting for this thread via an `_os_waitid()` call.

If successful, `_os_thexit` does not return to the caller.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l, mt_os_lib.l`

See Also

`_os_thfork`

_os_thfork

Fork a Thread

Syntax

```
#include <threads.h>
error_code _os_thfork(thread_t       *thread,
                      thread_attr_t  *attr,
                      void           **stack_top,
                      u_int32        *stack_size,
                      void           *func,
                      void           *arg,
                      void           *data);
```

Description

`_os_thfork()` forks a new thread of control in the current process.

`thread` is a pointer to a `thread_t` object which is used to return the newly created thread's ID.

`attr` is a pointer to a thread attribute structure. The thread attribute structure is declared in `threads.h`.

`stack_top` is a pointer to the top of user stack to be used by this thread—`stack_top` will be the initial stack pointer for the thread.

If the value pointed to by `stack_size` is non-zero, the thread's user stack will be allocated by the system and a pointer to the allocated memory will be returned in the location pointed to by `stack_top`. The size of the actual stack allocated will be returned at `stack_size`.

If `stack_size` points to zero, the value pointed to by `stack_top` will be used as the thread's stack.

`func` is the address of the initial function to be executed by the forked thread.

`arg` is passed as the only argument to the function specified by `func`.

`data` is the address of thread specific data. The value pointed to by `data` is maintained while the thread runs.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l, mt_os_lib.l`

See Also

`_os_thexit`
`_os_thread`

Syntax

```
#include <funcs.h>
#include <threads.h>
error_code _os_thread(u_int32   code,
                      thread_t  thread_id,
                      void      *arg);
```

Description

`_os_thread()` sets thread parameters for the thread specified by `thread_id`.

`code` is the subfunction to perform. The legal values for `code` are `TH_ORPHAN` to orphan the thread and `TH_TSDATA` to set the thread specific data for the thread.

If code is `TH_TSDATA`, `arg` is the thread specific data pointer.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l, mt_os_lib.l`

See Also

`_os_thexit`
`_os_thfork`

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_tlink(
     u_int32      trap_num,
     char         *mod_name,
     void         **lib_exec,
     mh_trap      **mod_head,
     void         *params,
     u_int32      mem_size);
```

Description

`_os_tlink()` attempts to link or load the specified module. If a trap module already exists for the specified trap number, an error is returned. If static storage is required for the trap handler, it is allocated and initialized.

`trap_num` specifies the user trap number (`1` through `15`).

`mod_name` is a pointer to the name of the trap module. If `mod_name` is zero or points to a null string, the trap handler is unlinked.

`lib_exec` is a pointer to the location where `_os_tlink()` stores the pointer to the trap execution entry point.

`mod_head` is a pointer to the location where `_os_tlink()` stores the pointer to the trap module.

`params` is a pointer to any additional parameters. Currently, this is not implemented.

`mem_size` specifies the additional memory size.

Note

OS-9 only allows trap handlers to run in system state. OS-9 for 68K allows trap handlers to run in either system state or user state.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_tlinkm()

Install User Trap Handling Module by Module Pointer

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_tlinkm(
     u_int32      trap_num,
     mh_com       *mod_head,
     void         **lib_exec,
     void         *params,
     u_int32      mem_size);
```

Description

`_os_tlinkm()` is passed a pointer to the module to install. If a trap module already exists for the specified trap number, an error is returned. If static storage is required for the trap handler, it is allocated and initialized.

`trap_num` specifies the user trap number (`1` through `15`).

`mod_head` is a pointer to the module header.

`lib_exec` is a pointer to the location where `_os_tlinkm()` stores the pointer to the trap execution entry point.

`params` is a pointer to the additional parameters, if necessary.

`mem_size` specifies the additional memory size.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_tranpn()

Translate User Path to System Path

Syntax

```
#include <io.h>
#include <types.h>
error_code _os_tranpn(
     process_id   proc_id,
     path_id      user_path,
     path_id      *sys_path);
```

Description

`_os_tranpn()` translates a user path number to a system path number.

`proc_id` specifies the process ID.

`user_path` specifies the user path to translate.

`sys_path` is a pointer to the location where `_os_tranpn()` stores the mapped system path.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_transadd()

Translate Memory Address

Syntax

```
#include <virtual.h>
#include <types.h>
error_code _os_transadd(
     u_int32      *size,
     u_int32      mode,
     void         **blk_addr,
     void         *reserved);
```

Description

`_os_transadd()` translates an address to or from its external bus address.

The size of the block to translate is passed at the location pointed to by `size`. `_os_transadd()` also stores the size of the block translated at the location pointed to by `size`.

`mode` specifies the type of translation:

- If `mode` is zero, the local CPU address is translated to the external bus address.
- If `mode` is one, the external bus address is translated to the local CPU address.

The address of the block to translate is passed at the location pointed to by `blk_addr`. `_os_transadd()` also stores the block that was translated at the location pointed to by `blk_addr`.

`_os_transadd()` is used when the external bus address must be passed to hardware devices, such as DMA-type controllers.

If the specified source block is nonlinear with respect to its destination mapping, `_os_transadd()` returns the maximum number of bytes accessible at the translated address.

`reserved` is a pointer reserved for future use.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os9_translate()

Translate Memory Address

Syntax

```
#include <memory.h>
#include <types.h>
error_code _os9_translate(
     u_int32      *size,
     u_int32      mode,
     void         **blk_addr);
```

Description

`_os9_translate()` translates an address to or from its external bus address.

The size of the block to translate is passed at the location pointed to by `size`. `_os9_translate()` also stores the size of the block translated at the location pointed to by `size`.

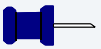
`mode` specifies the type of translation:

- If `mode` is zero, the local CPU address is translated to the external bus address.
- If `mode` is one, the external bus address is translated to the local CPU address.

The address of the block to translate is passed at the location pointed to by `blk_addr`. `_os9_translate()` also stores the block that was translated at the location pointed to by `blk_addr`.

`_os9_translate()` is used when the external bus address must be passed to hardware devices, such as DMA-type controllers.

If the specified source block is nonlinear with respect to its destination mapping, `_os9_translate()` returns the maximum number of bytes accessible at the translated address.



Note

`_os9_translate()` is only available if colored memory is enabled.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_uacct(
     u_int32     acct_code,
     pr_desc     *proc_desc);
```

Description

`_os_uacct()` provides a means for users to set up an accounting system. `_os_uacct()` is called by the kernel whenever it forks or exits a process. This provides a mechanism for users to keep track of system operators.

To install a handler for this service request, the `_os_setsvc()` system call must be used to add the user's accounting routine to the system's service request dispatch table. This is usually done in an OS9P2 module.

You may perform your own system accounting by calling `_os_uacct()` with a user defined `acct_code` identifying the operation to perform. For example, when the kernel forks a process it identifies the operation by passing the F_FORK code to the accounting routine.

`acct_code` is the operation identifier. This is usually a system call function code. Only the lower 16 bits of `acct_code` are used.

`proc_desc` is a pointer to the current process descriptor.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_unlink()

Unlink Module by Address

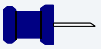
Syntax

```
#include <module.h>
#include <types.h>
error_code _os_unlink(mh_com *mod_head);
```

Description

`_os_unlink()` informs the system that the calling process no longer needs the module and decrements the module's link count.

`mod_head` is a pointer to the module header.



Note

Some modules, such as device drivers in use and all modules included in the bootfile, cannot be unlinked.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_unload()`

`F$UnLink`	***OS-9 for 68K Technical Manual***
`F_UNLINK`	***OS-9 Technical Manual***

_os_unload()

Unlink Module by Name

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_unload(
     char         *mod_name,
     u_int32      type_lang);
```

Description

`_os_unload()` locates the module specified by `mod_name` in the module directory and decrements its link count.

`mod_name` is a pointer to the module name.

`type_lang` specifies the module's type and language. Only the lower 16 bits of `type_lang` are used.

Library

`os_lib.l`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

See Also

`_os_unlink()`	
`F$UnLoad`	***OS-9 for 68K Technical Manual***
`F_UNLOAD`	***OS-9 Technical Manual***

_os_vmodul()

Verify Module

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_vmodul(
     mh_com       *mod_head,
     mh_com       *mod_block,
     u_int32      block_size);
```

Description

`_os_vmodul()` checks the module header parity and CRC bytes of an OS-9 module. If the header values are valid, the module is entered into the module directory. The current module directory is searched for another module with the same name. If a module with the same name and type exists, the module with the highest revision level is retained in the module directory. Ties are broken in favor of the established module.

`mod_head` is a pointer to the address of the module. `mod_block` is a pointer to the memory block containing the module. `block_size` specifies the size of the memory block containing the module.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_crc()`
`_os_load()`

_os9_vmodul()

Validate Module

Syntax

```
#include <module.h>
#include <types.h>
error_code _os9_vmodul(
     u_int32      mod_group,
     u_int32      size,
     mh_com       *mod_head,
     mod_dir      **dir_entry);
```

Description

`_os9_vmodul()` checks the module header parity and CRC bytes of an OS-9 module. If the header values are valid, the module is entered into the module directory, and a pointer to the directory entry is returned. The current module directory is searched for another module with the same name. If a module with the same name and type exists, the module with the highest revision level is retained in the module directory. Ties are broken in favor of the established module.

`mod_group` specifies the beginning of the module group (ID).

`size` is the size of the module.

`mod_head` is a pointer to the module.

`dir_entry` is a pointer to the location where `_os9_vmodul()` stores the pointer to the directory entry.

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_crc()`
`_os_load()`

_os_wait()

Wait for Child Process to Terminate

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_wait(
     process_id   *child_id,
     status_code  *status);
```

Description

`_os_wait()` deactivates the calling process until a child process terminates by executing an `_os_exit()` system call or is otherwise terminated. The child's ID number and exit status are returned to the parent. If the child process died due to a signal, status contains the signal code.

`child_id` is a pointer to the location where `_os_wait()` stores the process ID of the terminating child.

`status` is a pointer to the location where `_os_wait()` stores the child process' exit status code.

Note

If a signal is received by a process waiting for children to terminate, the process is activated. In this case, `child_id` contains zero, because no child process has terminated.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

_os_exit()
_os_fork()
_os_send()
F$Wait — *OS-9 for 68K Technical Manual*
F_WAIT — *OS-9 Technical Manual*

_os_waitid

Wait for a Specified Process or Thread to Exit

Syntax

```
#include <process.h>
error_code _os_waitid(process_id   *child_id,
                      error_code   *status,
                      signal_code  *signal);
```

Description

`_os_waitid()` causes the calling process or thread to wait for the specified thread or process.

The value pointed by `child_id` specifies the process or thread to wait for. If the value of `child_id` is the ID of a thread, the caller must be a thread in the same process as `child_id`. Otherwise an `EOS_IPRCID` error is returned.

If the call is successful, the status returned from `child_id` is returned in the location pointed to by `status`.

If the wait is interrupted by a signal, a value of `EOS_BSIG` is returned by `_os_waitid()` and the value of the signal that caused the interruption is stored in the location pointed to by `signal`.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l, mt_os_lib.l`

See Also

`_os_exit()`
`_os_thexit`
`_os_wait()`

Syntax

```
#include <lock.h>
#include <types.h>
error_code _os_waitlk(
     lk_desc     *lock,
     signal_code  *signal);
```

Description

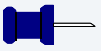
`_os_waitlk()` activates the next process waiting to acquire the lock. The next process in the lock's queue is activated and granted exclusive ownership of the resource lock. If no other process is waiting on the lock, the lock is simply marked free for acquisition. The calling process in either case is suspended and inserted into a waiting queue for the resource based on relative scheduling priority.

`lock` is a pointer to the lock on which to wait.

`signal` is a pointer to the location where `_os_waitlk()` stores the received signal.

If a process receives a signal while waiting on a lock, the process is activated without gaining ownership of the lock.

The process returns from the wait lock call with an `EOS_SIGNAL` error code and the signal code is returned through the signal pointer.



Note

If an `S_WAKEUP` signal is received by a waiting process, the signal code does not register and is zero.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`lock.l`

See Also

`_os_acqlk()`
`_os_caqlk()`
`_os_crlk()`
`_os_ddlk()`
`_os_dellk()`
`_os_rellk()`
`F_WAITLK`

OS-9 Technical Manual

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_write(
     path_id     path,
     void        *buffer,
     u_int32     *count);
```

Description

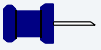
`_os_write()` outputs bytes to a file or device associated with the specified path number. The path must have been opened or created in the write or update access modes.

`path` is the specified path number for the file or device.

`buffer` is a pointer to the data buffer.

The number of bytes to write is passed at the location pointed to by count. `_os_write()` also stores the number of bytes written at the location pointed to by count.

Data is written to the file or device without processing or editing. If data is written past the present end-of-file, the file is automatically expanded.



Note

On RBF devices, any record that was locked is released.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

_os_open()
_os_create()
_os9_create()
_os_writeln()

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_writeln(
     path_id     path,
     void        *buffer,
     u_int32     *count);
```

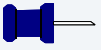
Description

`_os_writeln()` outputs bytes to a file or device associated with the specified path number. The path must have been opened or created in write or update access modes. `_os_writeln()` writes data until it encounters a carriage return character or `count` bytes. Line editing is also activated for character-oriented devices such as terminals and printers.

`path` is the path number of the file or device.

`buffer` is a pointer to the location where the bytes to be written are placed.

The number of bytes to write is passed at the location pointed to by `count`. `_os_writeln()` also stores the number of bytes written at the location pointed to by `count`.



Note

On RBF devices, any record that was locked is released.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_open()`
`_os9_create()`
`_os_create()`
`_os_write()`

_os_yield

Yield the Processor

Syntax

```
#include <process.h>
error_code _os_yield(void);
```

Description

`_os_yield()` causes the calling process or thread to be placed back into the active queue. The active queue contents are aged and the highest aged process is given control of the processor. In other words, `_os_yield()` causes the operating system to advance to the next executable process or thread. It is possible that the next executable process or thread will be the one that called `_os_yield()`. The status of the process' or thread's signal mask remains unchanged during this system call. `_os_yield()` is much like `_os_sleep()` with a tick count of 1, except that signals are not implicitly unmasked.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_sleep()`

outc()

Write Character to I/O Address

Syntax

```
#include <regs.h>
#include <types.h>
void outc(
     void         *address,
     u_int32      data);
```

Description

`outc()` writes a character to an I/O address on the 80x86 processors or memory address on all other processors.

`data` specifies the character to write. Only the lower 8 bits are used.

`address` specifies the I/O or memory address.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

See Also

`outl()`
`outw()`
`inc()`
`inl()`
`inw()`

outl()

Write Integer to I/O Address

Syntax

```
#include <regs.h>
#include <types.h>
void outl(
     void         *address,
     u_int32      data);
```

Description

`outl()` writes a long value to the I/O address on the 80x86 or the memory address on all other processors.

`data` specifies the long value to write.

`address` specifies the I/O or memory address.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

See Also

`outc()`
`outw()`
`inc()`
`inl()`
`inw()`

outw()

Write Word to I/O Address

Syntax

```
#include <regs.h>
#include <types.h>
void outw(
     void         address,
     u_int32      data);
```

Description

`outw()` writes a word to the I/O address on the 80x86 or the memory address on all other processors.

`data` specifies the word to write. Only the lower 16 bits are used.

`address` specifies the I/O or memory address.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

See Also

`outc()`
`outl()`
`inc()`
`inl()`
`inw()`

_parsepath()

Parse Disk File (RBF) Pathlist

Syntax

```
#include <strings.h>
int _parsepath(char *string);
```

Description

`_parsepath()` parses a disk file (RBF) pathlist. `_parsepath()` is useful for programs that must determine the validity of a pathlist name without actually creating or opening a file.

`string` is a pointer to the pathlist to parse.

`_parsepath()` returns the number of bytes in the valid pathlist. If RBF does not accept the pathlist as valid, `_parsepath()` returns `-1`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_prsnam()`	
`F$PrsNam`	***OS-9 for 68K Technical Manual***
`F_PRSNAM`	***OS-9 Technical Manual***

pause()

Wait for Signal

Syntax

```
#include <signal.h>
int pause(void);
```

Description

`pause()` suspends a task until a signal is received. `pause()` always returns `-1` unless the received signal caused the process to terminate. In this case, `pause()` never returns.

Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`kill()`	
`_os_sleep()`	
`_os9_sleep()`	
`sleep()`	
`F$Sleep`	***OS-9 for 68K Technical Manual***
`F_SLEEP`	***OS-9 Technical Manual***

pclose()

Close a Pipe from/to a Process

Syntax

```
#include <UNIX/os9def.h>
#include <stdio.h>
int pclose(FILE *stream)
```

Description

`pclose()` closes a pipe from or to a process that was opened by `popen()`. `pclose()` waits for the associated process to terminate and returns the exit status of the command.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

See Also

`popen()`

Syntax

```
#include <stdio.h>
void perror(const char *chptr);
```

Description

`perror()` maps the error number in `errno` to an error message.

`chptr` is a pointer to a character. If `chptr` is not a null pointer and the character pointed to by `chptr` is not a null character, `perror()` writes a sequence of characters to the standard error stream. The sequence written contains `chptr`, a colon (`:`), a space, and an appropriate error message string followed by a newline character.

The contents of the error message strings are the same as those returned by `strerror()` with parameter `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`clib.l`

See Also

`strerror()`

pffinit()

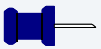
Initialize for Float Output (Obsolete)

Syntax

```
int pffinit(void);
```

Description

`pffinit()` was used on the 6809 C compiler to allow `printf()` to perform `float` and `double` conversions. A dummy function named `pffinit()` exists here for 6809/68000 portability.



Note

`pffinit()` is historical and may be removed in a future release.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

pflinit()

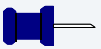
Initialize for Longs Output (Obsolete)

Syntax

```
int pflinit(void);
```

Description

`pflinit()` was used on the 6809 C compiler to allow `printf()` to perform `long int` conversions. A dummy function named `pflinit()` exists here for 6809/68000 portability.



Note

`pflinit()` is historical and may be removed in a future release.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

pipe()

Create an Interprocess Communication Channel

Syntax

```
#include <UNIX/os9def.h>
int pipe(unsigned int fd[2]);
```

Description

`pipe()` creates an I/O mechanism called a pipe and returns 2 path numbers, `fd[0]` and `fd[1]`. Both paths are open for `read` and `write` access. When the pipe is written, up to 4 KB of data are buffered before the writing process is blocked. Data read from one of the paths accesses data written to either of the paths on a FIFO (first-in-first-out) basis.

The standard programming model is that after the pipe has been set up, two (or more) cooperating processes (created by subsequent `_os_exec` calls) passes data through the pipe using `read` and `write` I/O calls.

If an error occurs, the `pipe`() returns `-1` and sets the global variable `errno` to indicate the error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

See Also

`pclose()`
`popen()`

popen()

Open a Pipe From/To a Process

Syntax

```
#include <UNIX/os9def.h>
#include <stdio.h>
FILE *popen(
     char         *command,
     char         *type);
```

Description

`popen()` opens a pipe from or to a process. The arguments to `popen()` are pointers to null-terminated strings containing, respectively, a shell command line and an I/O mode, either `r` for reading or `w` for writing. `popen()` creates a pipe between the calling process and the command to be executed. The value returned is a stream pointer such that you can write to the standard input of the command, if the I/O mode is `w`, by writing to the file stream. If the I/O mode was `r`, you can read from the standard output of the command by reading from the file stream. A stream opened by `popen()` should be closed with `pclose()` which waits for the associated process to terminate and returns the exit status of the command.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

See Also

`pclose()`
`pipe()`
`popen()`

pow()

Power Function

Syntax

```
#include <math.h>
double pow(
     double       x,
     double       y);
```

Description

pow() returns x raised to the y power. A domain error occurs, EDOM stored in errno, if x is negative. A domain error also occurs if both x and y are zero. A range error, ERANGE stored in errno, occurs on underflow and overflow. pow() returns HUGE_VAL for both errors.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

Possible Errors

EDOM
ERANGE

Syntax

```
#include <stdio.h>
int prerr(
     int          path,
     int          errnum);
```

Description

`prerr()` prints an error message corresponding to an error number on the standard error path.

`path` is the path number of the error message file.

`errnum` is the error number. It is divided into two numbers.

- **OS-9 for 68K:** Error numbers are 2 bytes wide. The most significant byte is the error's group number, and the least significant byte is the specific error within that group.
- **OS-9:** Error numbers are 4 bytes wide. The most significant 2 bytes are the error's group number, and the least significant 2 bytes are the specific error within that group.

`preer()` prints a message in the form "`Error #ggg:sss`" where `ggg` is the group number of the error and `sss` is the specific error number within the group. If path is non-zero, the opened error message file is searched for further text related to the error. If more text is available, it is printed.

Example

```
prerr(0, 216)                                    /*  function call */
Error #000:216                                   /*  output */
prerr(errmsg_path, 216)                          /*  function call */
Error #000:216  (E$PNNF) File not found.         /*  output line 1 */
The pathlist does not lead to any known file. /*  output line 2 */
```

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_perr()`
`F$PErr` ***OS-9 for 68K Technical Manual***

_prgname()

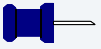
Get Module Name

Syntax

```
#include <module.h>
char *_prgname(void);
```

Description

`_prgname()` returns a pointer to the name of the calling module. Normally, the `argv[0]` string indicates the name of the program as called by the parent process with `os9exec()`. You can use `_prgname()` to determine the actual name of the module as it appears in the module directory.



Note

If the code that calls `_prgname()` is executing in a trap handler, the name of the trap handler module is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_errmsg()`
`os9exec()`

printf()

Formatted Output

Syntax

```
#include <stdio.h>
int printf(const char *format, ...);
```

Description

`printf()` is a standard C library function that performs formatted output to `stdout`. It converts, formats, and prints any parameters as indicated by the control string.

`format` is a pointer to a control string.

`printf()` returns the number of characters transmitted, or a negative value if an output error occurred.

Note

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `printf()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fprintf()`
`scanf()`
`sprintf()`

_prsnam()

Parse Path Name Segment

Syntax

```
#include <strings.h>
int _prsnam(char *string);
```

Description

`_prsnam()` parses a path name segment.

`string` is a pointer to the path name segment.

`_prsnam()` returns the number of characters in a valid path name segment. If the path name segment is invalid, `-1` is returned. Use `_parsenam()` to determine if a path is a valid disk file (RBF) pathlist.

You can use successive calls to `_prsnam()` to parse a complete path name.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_prsnam()`

`F$PrsNam`	***OS-9 for 68K Technical Manual***
`F_PRSNAM`	***OS-9 Technical Manual***

Syntax

```
#include <stdio.h>
int putc(
     int          chr,
     FILE         *stream);
int putchar(int chr);
```

Description

`putc()` converts a character to an `unsigned char`. It writes the character to the output stream at the position indicated by the associated file position indicator for the stream, if defined, and advances the indicator appropriately. If the stream cannot support positioning requests or if the stream was opened with append mode, the character is appended to the output stream.

If it is implemented as a macro, it may evaluate the stream more than once. Therefore, the parameter should never be an expression with side effects.

`putchar()` is similar to `putc()` with the second parameter `stdout`.

`chr` is the character to write. Only the lower 16 bits of `chr` are used.

`stream` is a pointer to the C I/O `FILE` structure.

`putc()` returns the character written. If a write error occurs, the error indicator for the stream is set and `putc()` returns `EOF`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

fopen()
fputc()
getc(), getchar()

putenv()

Change or Add Value to Environment

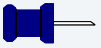
Syntax

```
#include <UNIX/os9def.h>
int putenv(char *string)
```

Description

`putenv()` sets the value of an environment variable. `string` is in the form "`name=value`" where `name` is the name of the environment variable to set and `value` is the new value for the environment variable. If the variable named exists, its value is changed to `value`. If the variable name does not exist, a new environment list entry is created and set to the value passed. `putenv`() makes a copy of `string` so subsequent modifications to `string` do not affect the environment.

`putenv`() returns non-zero if it was unable to obtain enough space using `malloc` for an expanded environment, otherwise zero.



Note

`putenv()` manipulates the environment pointed to by `_environ`, and can be used in conjunction with `getenv`(). However, `envp` (the third argument to main) is not changed. This routine uses `malloc` to enlarge the environment.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

puts()

Output String to File

Syntax

```
#include <stdio.h>
int puts(const char *strptr);
```

Description

`puts()` writes `strptr` to the stream pointed to by `stdout` and adds a newline character to the output. The terminating null character is not written.

`strptr` is a pointer to a string to write to `stdout`.

`puts()` returns `EOF` if an error occurs. Otherwise, it returns a non-negative value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fgets()`
`gets()`

putw()

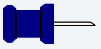
Output Word to File

Syntax

```
#include <stdio.h>
int putw(
     int          num,
     FILE         *stream);
```

Description

`putw()` writes an integer, converted to a word, to a file. `putw()` neither assumes nor causes any special alignment in the file. `num` is the integer to write to the structure `FILE`. `stream` is a pointer to the file. If successful, `putw()` returns the value it has written. Otherwise, it returns `-1`.



Note

`putw()`, like `getw()`, is machine-dependent because the size of the integer it outputs varies with the integer size of the machine on which it resides. This compiler defines `int` values as 4-byte quantities, but `putw()` actually outputs 2 bytes.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`getw()`

qsort()

Quick Sort

Syntax

```
#include <stdlib.h>
void qsort(
     void         *base,
     size_t       nmemb,
     size_t       size,
     int (*compar)(const void *, const void *));
```

Description

`qsort()` sorts an array. `base` is a pointer to the initial array element. `nmemb` is the number of elements in the array. `size` is the size of each array element. `compar` is a pointer to a comparison function. The contents of the array are sorted into ascending order according to `compar`. `compar` is called with two parameters that point to the objects being compared. `compar` returns an integer that is:

- Negative, if the first parameter is less than the second.
- Zero, if the first parameter is equal to the second.
- Positive, if the first parameter is greater than the second.

If two elements compare as equal, their order in the sorted array is unspecified. `qsort()` returns no value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

raise()

Send Signal To Program

Syntax

```
#include <signal.h>
int raise(int sig);
```

Description

`raise()` sends a signal to the executing program.

`sig` specifies the signal to send.

`raise()` returns zero if successful. Otherwise, it returns a non-zero value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

rand()

Return Random Integer

Syntax

```
#include <stdlib.h>
int rand(void);
```

Description

`rand()` computes a sequence of pseudo-random integers in the range zero to `RAND_MAX` and returns a pseudo-random integer.

`rand()` uses the following algorithm:

```
static unsigned long int next = 1;
int rand(void)      * RAND_MAX assumed to be 32767
{
   next = next * 1103515245 + 12345;
   return (unsigned int) (next/65536) % 32768;
}
```

`next` contains the current internal value for calculating pseudo-random numbers. Because the value of `next` is divided by 65536 before returning the value, this algorithm avoids the common problem of non-random, low-order bits.

By storing the value in an `int`, this algorithm also allows the period of the sequence to be greater than `RAND_MAX`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <modes.h>
int read(
     int          path,
     char         *buffer,
     unsigned     count);
```

Description

`read()` reads bytes from a path.

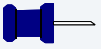
`path` specifies the path number of the path from which to read. The path number is an integer indicating one of the standard path numbers (0, 1, or 2) or a path number returned from a successful call to `open()`, `creat()`, `create()`, or `dup()`.

`buffer` is a pointer to a space with at least `count` bytes of memory into which `read()` puts the data read from the path.

`count` is the minimum buffer size. It is guaranteed that at most, `count` bytes are read, but often less are read, either because the path serves a terminal and input stops at the end of a line or the end-of-file has been reached.

`read()` essentially performs a **raw** `read`; that is, the `read` is performed without translating characters. The characters are passed to the program as read.

`read()` returns the number of bytes actually read (`0` indicates the end of the file). If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

End-of-file is returned as zero bytes being read, not an error indication.

Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`creat()`
`create()`
`dup()`
`open()`
`read()`
`readln()`

`I$Read`	***OS-9 for 68K Technical Manual***
`I_READ`	***OS-9 Technical Manual***

Syntax

```
#include <dir.h>
struct direct *readdir(DIR *dirp);
```

Description

`readdir()` returns a pointer to a structure containing the next directory entry.

`dirp` is a pointer to the directory.

`readdir()` returns `NULL` upon reaching the end of the directory or detecting an invalid `seekdir()` operation.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

`closedir()`
`opendir()`
`rewinddir()`
`seekdir()`
`telldir()`

readln()

Read Bytes from Path

Syntax

```
#include <modes.h>
int readln(
     int          path,
     char         *buffer,
     unsigned     count);
```

Description

`readln()` reads bytes from a path.

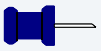
`path` specifies the path number of the file from which to read. The path number is an integer indicating one of the standard path numbers (`0`, `1`, or 2) or a path number returned from a successful call to `open()`, `creat()`, `create()`, or `dup()`.

`buffer` is a pointer to a space with at least `count` bytes of memory into which `read()` puts the data read from the path.

`count` is the minimum buffer size. It is guaranteed that at most, `count` bytes are read. Often less are read, either because the path serves a terminal and input stops at the end of a line or the end-of-file has been reached.

`readln()` causes line-editing, such as echoing, to take place and returns once a `\n` is read in the input or the number of bytes requested has been read. `readln()` is the preferred call for reading from the terminal.

`readln()` returns the number of bytes actually read (`0` indicating end of file). If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

End-of-file is returned as zero bytes being read, not an error indication.

Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`creat()`
`create()`
`dup()`
`open()`
`_os_readln()`
`read()`

`I$ReadLn`	***OS-9 for 68K Technical Manual***
`I_READLN`	***OS-9 Technical Manual***

Syntax

```
#include <UNIX/os9def.h>
int readv(
     unsigned int fd,
     struct iovec *iov,
     unsigned int iovcnt);
```

Description

`readv()` reads input, the same action as `read()`. The difference is that `readv()` scatters the input data into the `iovcnt` buffers specified by the members of the `iov` array:

```
iov[0], iov[1], ...iov[iovcnt-1].
```

If `iovcnt` is zero, the value of the `iov_len` member in an `iovec` structure is less than zero, or all the `iov_len` members of the `iovec` structures are zero, `readv()` returns `-1` and sets the global variable `errno`.

For `readv()`, the `iovec` structure is defined as:

```
struct iovec{
     caddr_t iov_base;
     int iov_len
};
```

Each `iovec` entry specifies the base address and length of an area in memory where data should be placed. `readv()` always fills an area completely before proceeding to the next.

Upon successful completion, `readv()` returns the number of bytes actually read and placed in the buffers.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

realloc()

Resize Block of Memory

Syntax

```
#include <stdlib.h>
void *realloc(
     void         *ptr,
     size_t       size);
```

Description

`realloc()` changes the size of a memory block. The contents of the object are unchanged up to the lesser of the new and old sizes. If the new size is larger, the value of the newly allocated portion of the object is indeterminate.

`ptr` is a pointer to the old block of memory. If `ptr` is a null pointer, `realloc()` behaves like `malloc()` for the specified size. Otherwise, if `ptr` does not match a pointer earlier returned by `calloc()`, `malloc()`, or `realloc()`, or if the space has been deallocated by a call to `free()`, `_lfree()`, `realloc()`, or `_lrealloc()`, the behavior is undefined.

`size` is the size of the new block of memory.

If the space cannot be allocated, `ptr` is unchanged.

If `size` is zero and `ptr` is not a null pointer, the object `ptr` points to is freed.

`realloc()` returns either a null pointer or a pointer to the possibly moved allocated space.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

calloc()
free()
_lcalloc()
_lfree()
_lrealloc()
malloc()
realloc()

Syntax

```
#include <UNIX/regexp.h>
regexp *regcomp(char *str);
```

Description

`regcomp()` compiles a string (`str`) into an internal form suitable for pattern matching with `regex()`. `regex()` matches a compiled regular expression against a string.

`regcomp()` returns a pointer to the compiled regular expression. Otherwise a string containing an error message is returned. If an error occurs, the function `regerror()` is called with a character string for the error and `NULL` is returned.

When the compiled regular expression is no longer needed, it can be passed to `free()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

See Also

`regerror()`
`regex(), regexec()`

regerror()

Regular Expression Error Handler

Syntax

```
#include <UNIX/regexp.h>
void regerror (char *msg);
```

Description

`regerror()` is called by the functions `regex()` and `regcomp()` when various errors occur. `regerror()` prints the string "`regexp: `" and the string pointed to by `msg`.

If printing a message is inappropriate for the application using regular expressions, the programmer may define a `regerror()` function to override the `unix.l` version.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

See Also

`regcomp()`
`regex(), regexec()`

Syntax

```
#include <UNIX/regexp.h>
int regex(
     regexp       *re,
     char         *str);
```

Description

`regex()` compares a string (`str`) with a compiled regular expression (`re`).

`regex()` returns one if the strings match and zero if the strings don't match. If an error occurs, `regerror()` is called with the appropriate error message and zero is returned to indicate the strings did not match.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

See Also

`regcomp()`
`regerror()`

remove()

Remove File

Syntax

```
#include <stdio.h>
int remove(const char *filename);
```

Description

`remove()` deletes a file. Subsequent attempts to open a file using that name fail, unless the file is recreated.

`filename` is a pointer to the file to delete.

If the file is open, `remove()` returns `-1` with `errno` set to `EOS_SHARE`.

`remove()` returns zero if the operation succeeds. Otherwise, it returns a non-zero value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

rename()

Rename File

Syntax

```
#include <stdio.h>
int rename(
     const char   *old,
     const char   *new);
```

Description

`rename()` renames a file.

`old` is a pointer to the name of an existing file.

`new` is a pointer to the new name for the file.

The file named `old` is no longer accessible by that name. If a file named `new` exists prior to the call to `rename()`, it is overwritten.

`rename()` returns zero if the operation succeeds. Otherwise, it returns a non-zero value and the file name does not change.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

rewind()

Return File Pointer to Zero

Syntax

```
#include <stdio.h>
void rewind(FILE *stream);
```

Description

`rewind()` sets the file position indicator for the stream pointed to by `stream` to the beginning of the file. It is equivalent to the following except that the error indicator for the stream is also cleared:

```
(void)fseek(stream, 0L, SEEK_SET)
;
```

`stream` is a pointer to the C I/O `file` structure.

`rewind()` returns no value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fseek()`

rewinddir()

Reset Position of Directory Stream

Syntax

```
#include <dir.h>
void rewinddir(DIR *dirp);
```

Description

`rewinddir()` resets the position of the named directory stream to the beginning of the directory.

`dirp` is a pointer to the directory stream.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`closedir()`
`opendir()`
`readdir()`
`seekdir()`
`telldir()`

Syntax

```
#include <strings.h>
char *rindex(
     char         *ptr,
     int          ch);
```

Description

`rindex()` returns a pointer to the last occurrence of a character in a string. If the character is not found, `rindex()` returns `NULL`.

`ptr` is a pointer to the string to search.

`ch` is the character for which to search. Only the lower 8 bits of `ch` are used.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

Example

The following example code fragment looks for a period (`.`) in the string and sets `ptr` to point to it. The search starts from the end of the string and progresses to the front of the string. Note that `ch` is a character, not a pointer to a character:

```
func()
{
   char *ptr,*string;
   if((ptr = rindex(string,'.')) {
```

```
        process(ptr);
    } else {
        printf("No '.' found!\n");
    }
}
```

See Also

index()

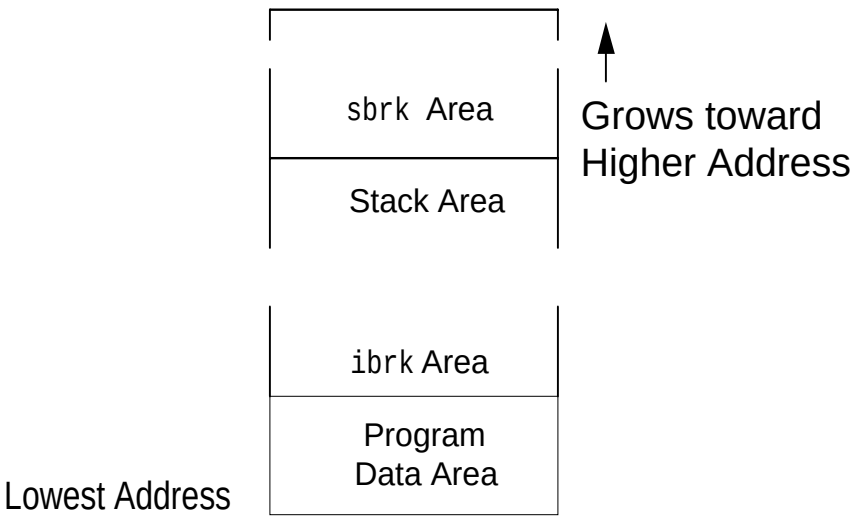
Syntax

```
#include <stdlib.h>
char *sbrk(unsigned size);
```

Description

`sbrk()` allocates memory from the top of the data area upwards.

Table 2-24 sbrk() Memory Allocation Table



`size` specifies the size of the memory block.

`sbrk()` grants memory requests by calling the `F$MEM` (for OS-9 for 68K) or the `F_MEM` (for OS-9) system call. This method resizes the data area to a larger size. The new memory granted is contiguous with the end of the previous data memory.

On systems without an MMU, `sbrk()` fails quickly because it may keep growing in size until the data area reaches other allocated memory. At this point, it is impossible to increase in size and an error is returned. A program may increase its data size by only 20K, even if 200K is available elsewhere.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

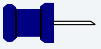
`_os_mem()`	
`_os_srqmem()`	
`F$Mem`	***OS-9 for 68K Technical Manual***
`F_MEM`	***OS-9Technical Manual***

Syntax

```
#include <stdio.h>
int scanf(const char *format, ...);
```

Description

`scanf()` reads input from `stdin`. `format` is a pointer to the control string. `scanf()` returns `EOF` if an input failure occurs before any conversion. Otherwise, `scanf()` returns the number of input items assigned, which can be fewer than provided for, or zero in the event of an early matching failure.



Note

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `scanf()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fscanf()`
`sscanf()`

sched_yield()

Yield Processor

Syntax

```
#include <sched.h>
int sched_yield(void);
```

Description

`sched_yield()` gives up a thread's timeslice, allowing the kernel to reschedule the active threads. If there are other active threads at the same priority as the calling thread, one of them will be given the processor. The standard aging process occurs at every `sched_yield()` call.

`sched_yield()` works much like `_os_sleep()` with a `ticks` value of 1, except that it does not implicitly unmask signals.

If successful, 0 is returned, otherwise an error number is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_sleep()`
`tsleep()`

Example

```
while (not_ready)
   sched_yield();/* allow other threads to run */
```

seekdir()

Set Position of Next Readdir

Syntax

```
#include <dir.h>
void seekdir(
     DIR          *dirp,
     long         loc);
```

Description

`seekdir()` sets the position of the next `readdir()` operation on the directory stream to `loc`. `dirp` is a pointer to the directory. `loc` is the location for the next `readdir()` operation. `loc` is a specific position in the directory stream returned by a previous call to `telldir()`.

Values returned by `telldir()` are valid only for the lifetime of the associated `dirp` pointer. If the directory is closed and then reopened, the `telldir()` value may no longer be valid. It is safe to use a previous `telldir()` value immediately after a call to `opendir()` and before any calls to `readdir()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`closedir()`
`opendir()`
`readdir()`
`rewinddir()`
`telldir()`

Syntax

```
#include <UNIX/os9types.h>
#include <types.h>
int select (
     unsigned int width,
     fd_set       *readfds,
     fd_set       *writefds,
     fd_set       *exceptfds,
     struct timeval *timeout);
```

Description

`select()` examines the path sets whose addresses are passed in `readfds`, `writefds`, and `exceptfds` to see if some of their paths are ready for reading, ready for writing, or have an exceptional condition pending. `width` is the number of bits to be checked in each bit mask that represent a path. The paths from `0` through `width-1` in the descriptor sets are examined. `width` is in the range `1-FD_SETSIZE`. The total number of ready paths in all the sets is returned. The macros `FD_SET`, `FD_CLR`, `FD-ISSET`, and `FD_ZERO` have been defined in `os9types.h` to manipulate these `fd_set` structures.

If `timeout` is not a NULL pointer, it specifies a maximum interval to wait for the selection to complete. If `timeout` is a NULL pointer, the `select` blocks indefinitely. To effect a poll, the `timeout` argument should be a non-NULL pointer, pointing to a zero-valued `timeval` structure.

Any `readfds`, `writefds`, and `exceptfds` may be given as NULL pointers if no paths are of interest.

`select()` returns a non-negative value on success. A positive value indicates the number of ready paths in the path sets. Zero indicates that the time limit referred to by `timeout` has expired. On failure, `select()` returns `-1`, sets `errno` to indicate the error, and the path sets are not changed.

Note

User applications using `select()` can not use the `_os_intercept()` function.

Note

No paths in the `exceptfds` are ever returned since OS-9 for 68K does not support the concept of “exceptional condition.”

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

set<name>()

Set Value of SPR, DCR, PREG, or TB

Syntax

```
#include <getset.h>
#include <types.h>
void _set_<name>(u_int32 val);
```

Description

The `_set_<name>()` class of functions can be used to set the value of Special Purpose Registers (SPRs), Device Control Registers (DCRs), Processor Registers (PREG), and Time-Based Registers (TBs).

Note

All functions are provided. It is the programmer's responsibility to avoid using system-state only registers from user-state and accessing registers that do not exist on the target processor. All registers are accessible from system-state. Only those noted are available from user-state. Attempting to use system state registers from user state causes unpredictable results.

Table 2-25 _set_<name>() Valid Name Values

<name>	Register Type	Processor(s) 403	505	601	603	User	Readable
bar	SPR		•				•
besr	DCR	•					•

Table 2-25 _set_<name>() Valid Name Values (continued)

<name>	Register Type	Processor(s) 403	505	601	603	User	Readable
br0	DCR	•					•
br1	DCR	•					•
br2	DCR	•					•
br3	DCR	•					•
br4	DCR	•					•
br5	DCR	•					•
br6	DCR	•					•
br7	DCR	•					•
cmpa	SPR		•				•
cmpb	SPR		•				•
cmpc	SPR		•				•
cmpd	SPR		•				•
cmpe	SPR		•				•
cmpf	SPR		•				•
cmpg	SPR		•				•
cmph	SPR		•				•
counta	SPR		•				•

Table 2-25 _set_<name>() Valid Name Values (continued)

<name>	Register Type	Processor(s) 403	505	601	603	User	Readable
countb	SPR		•				•
cr	PREG	•	•	•	•	•	•
ctr	SPR	•	•	•	•	•	•
dabr	SPR			•			•
dac1	SPR	•					•
dac2	SPR	•					•
dar	SPR		•	•	•		•
dbat0l	SPR				•		•
dbat0u	SPR				•		•
dbat1l	SPR				•		•
dbat1u	SPR				•		•
dbat2l	SPR				•		•
dbat2u	SPR				•		•
dbat3l	SPR				•		•
dbat3u	SPR				•		•
dbcr	SPR	•					•
dbsr	SPR	•					•

Table 2-25 _set_<name>() Valid Name Values (continued)

<name>	Register Type	Processor(s) 403	505	601	603	User	Readable
dccr	SPR	•					•
dcmp	SPR				•		•
dec	SPR		•	•	•		•
der	SPR		•				•
dmacc0	DCR	•					•
dmacr0	DCR	•					•
dmacr1	DCR	•					•
dmacr2	DCR	•					•
dmacr3	DCR	•					•
dmact0	DCR	•					•
dmact1	DCR	•					•
dmact2	DCR	•					•
dmact3	DCR	•					•
dmada0	DCR	•					•
dmada1	DCR	•					•
dmada2	DCR	•					•
dmada3	DCR	•					•

Table 2-25 _set_<name>() Valid Name Values (continued)

<name>	Register Type	Processor(s) 403	505	601	603	User	Readable
dmasa0	DCR	•					•
dmasa1	DCR	•					•
dmasa2	DCR	•					•
dmasa3	DCR	•					•
dmasr	DCR	•					•
dmiss	SPR				•		•
dpdr	SPR		•				•
dsisr	SPR		•	•	•		•
ear	SPR			•	•		•
ecr	SPR		•				•
eid	SPR		•				•
eie	SPR		•				•
esr	SPR	•					•
evpr	SPR	•					•
exier	DCR	•					•
exisr	DCR	•					•
fpecr	SPR		•				•

Table 2-25 _set_<name>() Valid Name Values (continued)

<name>	Register Type	Processor(s) 403	505	601	603	User	Readable
fpscr	PREG	•	•	•	•	•	•
hash1	SPR				•		•
hash2	SPR				•		•
hid0	SPR			•	•		•
hid1	SPR			•			•
hid15	SPR			•			•
hid2	SPR			•			•
hid5	SPR			•			•
iabr	SPR			•	•		•
iac1	SPR	•					•
iac2	SPR	•					•
ibat0l	SPR			•	•		•
ibat0u	SPR			•	•		•
ibat1l	SPR			•	•		•
ibat1u	SPR			•	•		•
ibat2l	SPR			•	•		•
ibat2u	SPR			•	•		•

Table 2-25 _set_<name>() Valid Name Values (continued)

<name>	Register Type	Processor(s) 403	505	601	603	User	Readable
ibat3l	SPR			•	•		•
ibat3u	SPR			•	•		•
icadr	SPR		•				•
iccr	SPR	•					•
iccst	SPR		•				•
icdat	SPR		•				•
icmp	SPR				•	•	•
ictrl	SPR		•				•
imiss	SPR				•		•
iocr	DCR	•					•
lctrl1	SPR		•				•
lctrl2	SPR		•				•
lr	SPR	•	•	•	•	•	•
mq	SPR			•		•	•
nri	SPR		•				•
msr	PREG	•	•	•	•		•
pbl1	SPR	•					•

Table 2-25 _set_<name>() Valid Name Values (continued)

<name>	Register Type	Processor(s) 403	505	601	603	User	Readable
pbl2	SPR	•					•
pbu1	SPR	•					•
pbu2	SPR	•					•
pir	SPR			•			•
pit	SPR	•					•
rpa	SPR				•		•
rtcl	SPR			•			•
rtcu	SPR			•			•
sdr1	SPR			•	•		•
sprg0	SPR	•		•	•		•
sprg1	SPR	•		•	•		•
sprg2	SPR	•		•	•		•
sprg3	SPR	•		•	•		•
sr0	PREG			•	•		•
sr1	PREG			•	•		•
sr2	PREG			•	•		•
sr3	PREG			•	•		•

Table 2-25 _set_<name>() Valid Name Values (continued)

<name>	Register Type	Processor(s) 403	505	601	603	User	Readable
sr4	PREG			•	•		•
sr5	PREG			•	•		•
sr6	PREG			•	•		•
sr7	PREG			•	•		•
sr8	PREG			•	•		•
sr9	PREG			•	•		•
sr10	PREG			•	•		•
sr11	PREG			•	•		•
sr12	PREG			•	•		•
sr13	PREG			•	•		•
sr14	PREG			•	•		•
sr15	PREG			•	•		•
srr0	SPR	•	•	•	•		•
srr1	SPR	•	•	•	•		•
srr2	SPR	•					•
srr3	SPR	•					•
tbhi	SPR	•					•

Table 2-25 _set_<name>() Valid Name Values (continued)

<name>	Register Type	Processor(s) 403	505	601	603	User	Readable
tbl	SPR		•		•		
tblo	SPR	•					•
tbu	SPR		•		•		
tcr	SPR	•					•
tsr	SPR	•					•
xer	SPR	•	•	•	•	•	•

Attributes

Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library

cpu.l

See Also

get<name>()

setbuf()

Fix File Buffer

Syntax

```
#include <stdio.h>
void setbuf(
     FILE         *stream,
     char         *buf);
```

Description

`setbuf()` sets up buffering for an I/O stream. It may be used only after the stream has been associated with an open file and before any other operation is performed on the stream. It is similar to `setvbuf()` called with `_IOFBF` for `mode` and `BUFSIZ` for `size`, or, if `buf` is a null pointer, with `_IONBF` for `mode`.

`stream` is a pointer to the C I/O `FILE` structure.

`buf` is a pointer to the file's buffer.

`setbuf()` returns no value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fopen()`
`getc(), getchar()`
`putc(), putchar()`
`setvbuf()`

_setcrc()

Re-Validate Module CRC

Syntax

```
#include <module.h>
int _setcrc(mh_com *module);
```

Description

`_setcrc()` updates the header parity and CRC of a module in memory. The module must have the correct size and sync bytes; other parts of the module are not checked.

`module` is a pointer to the module to update.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

WARNING

Although `_os_setcrc()` and `_setcrc()` functions do update the header and CRC of a module in memory, the copy of the module header in the module directory is not affected. Therefore, you have an invalid module directory entry for the memory module with the new CRC and an error 236 results if links to the module are attempted.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_setcrc()`	
F$SetCRC	***OS-9 for 68K Technical Manual***
F_SETCRC	***OS-9 Technical Manual***

set_excpt_base()

Set Exception Table Base Address

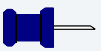
Syntax

```
#include <regs.h>
void set_excpt_base(u_int32 *dsreg);
```

Description

`set_excpt_base()` sets the exception table base address. It is a system state only system call.

`dsreg` is a pointer to the base address of the exception table.



Note

`set_excpt_base()` is supported only on the 80x86 family of processors version of the compiler.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

set_gdtr()

Set Global Descriptor Pointer

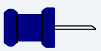
Syntax

```
#include <regs.h>
void set_gdtr(u_int32 *base);
```

Description

`set_gdtr()` sets the global descriptor pointer. It is a system state only system call.

`base` is a pointer to the global descriptor.



Note

`set_gdtr()` is supported only on the 80x86 family of processors version of the compiler.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

setgid()

Set Global Descriptor Pointer

Syntax

```
#include <UNIX/os9def.h>
int setgid(unsigned int gid);
```

Description

`setgid()` sets the user's group identification of the current process as specified by `gid`. `setgid()` returns `-1` on error and sets the global variable `errno` to indicate the error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

setime()

Set System Time

Syntax

```
#include <time.h>
int setime(struct sgtbuf *timebuf);
```

Description

`setime()` sets the system time from a time buffer. The time units are defined in the `time.h` header file. This function uses the year 1900 as a base, and the year passed to the function must be an offset of 1900. For example, to use the `setime()` function and set to the year 2011, you need to pass `111` in the year member of the `tm` structure.

`timebuf` is a pointer to the time buffer.

If successful, `setime()` returns a pointer to a buffer. Otherwise, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`getime()`	
`_os_setime()`	
`_os9_setime()`	
`F$STime`	***OS-9 for 68K Technical Manual***
`F_STIME`	***OS-9Technical Manual***

Syntax

```
#include <setjmp.h>
int setjmp(jmp_buf env);
```

Description

`setjmp()` is a macro which saves its calling environment in its `env` parameter for later use by a call to `longjmp()`.

`env` is a program environment structure.

If the return is from a direct call, `setjmp()` returns zero.

If the return is from a call to `longjmp()`, `setjmp()` returns a non-zero value.

`setjmp()` may only be used as follows:

- The entire expression of an expression statement. This may be cast to `void`. For example:

  ```
  (void) setjmp(env);
  ```

- The entire controlling expression of a selection or iteration statement. For example:

  ```
  while(setjmp(env))
  ```

- One expression of a relational or equality operator with the operand an integral constant expression, with the resulting expression being the entire controlling expression of a selection or iteration statement. For example:

  ```
  while(1<=setjmp(env))
  ```

- The operand of the unary operator (!) with the resulting expression being the entire controlling expression of a selection or iteration statement. For example:

  ```
  while(!setjmp(env))
  ```

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`longjmp()`

Syntax

```
#include <locale.h>
char *setlocale(
     int          category,
     const char   *locale);
```

Description

`setlocale()` selects the appropriate portion of the program's locale as specified by `category` and `locale`. You can use `setlocale()` to change or query the program's entire current locale or portions of it.

`category` can be one of the following values:

Table 2-26 setlocale() Category Values

Name	Description
`LC_ALL`	Names the program's entire locale. The other values for `category` name only a portion of the program's locale.
`LC_COLLATE`	Affects the behavior of `strcoll()` and `strxfrm()`
`LC_CTYPE`	Affects the behavior of the character handling functions and the multibyte functions
`LC_MONETARY`	Affects the monetary formatting information returned by `localeconv()`
`LC_NUMERIC`	Affects the decimal point character for the formatted I/O functions and the string conversion functions, as well as the non-monetary formatting information returned by `localeconv()`
`LC_TIME`	Affects the behavior of `strftime()`

Currently, the C locale is fully supported.

If `locale` is `C`, the minimal environment for C translation is specified.

If `locale` is "", the value of the corresponding environment variable, `LC_TIME` for category `LC_TIME`, and so forth, is checked for validity. If the environment variable contains a valid locale string, `locale` is set accordingly. If it does not contain a valid string, `setlocale()` behaves as if passed an invalid locale string.

If the environment variable is not set or is set to the empty string, the existence and validity of the environment variable `LANG` is checked and `locale` is set from these. If `LANG` does not exist, `locale` is set to `C`.

At program startup, the equivalent of the following is executed:

```
setlocale(LC_ALL, "C");
```

If `locale` is a pointer to a string and the selection can be honored, `setlocale()` returns a pointer to the string associated with the specified category for the new locale. If the selection cannot be honored, `setlocale()` returns a null pointer and the program's locale is not changed.

A null pointer for `locale()` causes `setlocale()` to return a pointer to the string associated with the category for the program's current locale; the program's locale is not changed.

The pointer to the string returned by `setlocale()` is such that a subsequent call with that string value and its associated category restores that part of the program's locale. The program does not modify the string, but it may be overwritten by a subsequent call to `setlocale()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

atof()
atoi()
atol()
fprintf()
fscanf()
isalnum()
isalpha()
iscntrl()
isdigit()
isgraph()
islower()
isprint()
ispunct()
isspace()
isupper()
isxdigit()
_issjis1()
_issjis2()
_isjis_kana()
mblen()
mbstowcs()
mbtowc()
printf()
scanf()
sprintf()
sscanf()
vfprintf()
vprintf()
vsprintf()
wctomb()
wcstombs()
strtod()
strtol()
strtoul()
strcoll()
strftime()
strxfrm()

setpr()

Set Process Priority

Syntax

```
#include <process.h>
int setpr(
     int          pid,
     int          prior);
```

Description

`setpr()` sets the priority of a process. The lowest priority is zero; the highest is 65535.

`pid` is the process ID.

`prior` specifies the priority. Only the lower 16 bits of `prior` are used.

If an error occurs, such as the process not having the same user ID as the caller, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

setpwent()

Reset Password File

Syntax

```
#include <UNIX/pwd.h>
int setpwent(void);
```

Description

`setpwent()` has the effect of rewinding the password file to allow for repeated searches.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

See Also

`endpwent()`
`fgetpwent()`
`getpwent()`
`getpwnam()`
`getpwuid()`

setstat()

Set File Status

Syntax

```
#include <sg_codes.h>
#include <sgstat.h>
int setstat(
     int          code,
     int          path,
     char         *buffer);       /* code = 0* /
int setstat(
     int          code,
     int          path,
     long         size);          /* code = 2* /
```

Description

`setstat()` sets the path options or the file size of the file open on `path`.

If `code` is 0, the buffer is copied to the path descriptor options section. The `sgstat.h` header file contains the definitions for the path options.

If `code` is 2, `size` should be an `int` specifying the new file size. Only the lower 16 bits of `code` are used.

`path` is the path number of the open file.

`buffer` is a pointer to the buffer containing the path descriptor options.

`size` is the size of the new file.

If an error occurs, both forms of the call return `-1` and place the appropriate error code in the global variable `errno`.

Note

This call exists for 6809 portability. The `_os_ss` functions are the preferred versions of these function calls.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`getstat()`
`_os_ss_attr()`
`_os9_ss_close()`
`_os_ss_dcon()`
`_os_ss_dcoff()`
`_os_ss_dsrts()`
`_os_ss_enrts()`
`_os_ss_erase()`
`_os_ss_fd()`
`_os_ss_lock()`
`_os_ss_popt()`
`_os_ss_relea()`
`_os_ss_reset()`
`_os_ss_rfm()`
`_os_ss_sendsig()`
`_os_ss_size()`
`_os_ss_skip()`
`_os_ss_ticks()`
`_os_ss_wfm()`
`_os_ss_wtrack()`
`_os9_ss_open()`
`I$SetStt`

OS-9 for 68K Technical Manual

_setsys()

Set/Examine System Global Variables

Syntax

```
#include <sysglob.h>
#include <setsys.h>
int _setsys(
     int          glob,
     int          size[,
     int          value]);
```

Description

`_setsys()` changes or examines a system global variable. These variables are defined in `sys.l`. The same values are defined in the `setsys.h` header file for C program use. These variables begin with a `D_` prefix.

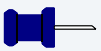
`glob` is the offset to the intended variable. Only the lower 16 bits of `glob` are used.

`size` is the size of the variable. `size | 0x80000000` is used to examine the variable.

`value` is an optional parameter used only when changing a variable.

`_setsys()` returns the value of the variable on an examination request. On a change request, `_setsys()` returns the value of the variable before the change.

If the examine or change request fails, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

Use `_getsys()` to examine variables without the possibility of accidental change.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_getsys()`
`_os_getsys()`
`_os_setsys()`
`F$SetSys` ***OS-9 for 68K Technical Manual***

setuid()

Set User ID

Syntax

```
#include <process.h>
int setuid(int uid);
```

Description

`setuid()` sets the group/user ID of the process to `uid`. The following restrictions govern the use of `setuid()`:

- User number `0.0` may change his/her ID to anything without restriction.
- A primary module owned by user `0.0` may change the process' user ID to anything without restriction.
- Any primary module may change the process' user ID to match the module's owner.

If the call fails, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`getuid()`

Syntax

```
#include <stdio.h>
int setvbuf(
     FILE         *stream,
     char         *buf,
     int          mode,
     size_t       size);
```

Description

`setvbuf()` sets up buffering for an I/O stream. It may be used only after the `stream` has been associated with an open file and before any other operation is performed on the stream.

`stream` is a pointer to the C I/O `FILE` structure.

`buf` is a pointer to a buffer. If `buf` is not a null pointer, the library may use the array it points to instead of a buffer allocated by `setvbuf()`.

`mode` determines how `stream` is buffered:

Table 2-27 setvbug() Parameter Mode Values

Mode	Description
`_IOFBF`	I/O is fully buffered
`_IOLBF`	I/O is line buffered
`_IONBF`	I/O is unbuffered

`size` specifies the array size. The contents of the array at any time are indeterminate.

`setvbuf()` returns zero if successful. If an invalid value is given for mode or if the request cannot be honored, `setvbuf()` returns a non-zero value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <signal.h>
int sigmask(int level);
```

Description

`sigmask()` controls the process' signal mask. A signal mask is an internal variable in each process descriptor which determines the signal handling for the process.

`level` specifies the signal level:

- `0` specifies clear
- `1` specifies increment
- `-1` specifies decrement

The process' signal level is not decremented below zero.

If a signal is received by a process whose signal mask is zero, normal program flow is interrupted and the signal is processed by executing the program's intercept routine.

If a signal is received by a process whose signal mask is non-zero, the signal is placed in a queue of signals waiting to be processed. The queued signals become active only when the process' signal mask becomes zero.

Regardless of the signal mask value, a process wakes up from event waits and I/O operation when signals arrive, but the signal handling routine is not executed until the signals are unmasked.

The process' signal mask is automatically incremented while the intercept routine is executing. This prevents the intercept routine from being accidentally re-entered if a new signal arrives. The process may use `sigmask()` within its intercept routine to allow re-entrant signals or to force the signal mask to remain non-zero when normal program execution resumes.

When a process makes an `F$Sleep` or `F$Wait` (for OS-9 for 68K) or `F_SLEEP` or `F_WAIT` (for OS-9) system call, its signal mask is automatically cleared. If any signals are pending, the process returns to the intercept routine without sleeping.

The `S_KILL` and `S_WAKE` signals ignore the state of the signal mask and are never queued. `S_KILL` terminates the receiving process, and `S_WAKE` ensures that the receiving process is active.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`. If errors do not occur, `sigmask()` returns zero.

Note

I/O operations using the `cio` library should not be performed by both the main program and the intercept routine.

If an intercept routine is exited with `longjmp()`, the signal mask is still set to one. Generally, the destination of the long jump should unmask signals immediately.

OS-9 for 68K: The depth to which signals may be queued is limited only by available memory.

OS-9: There is a set limit.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

_os_intercept()	
_os_send()	
_os_sigmask()	
F$SigMask	***OS-9 for 68K Technical Manual***
F_SIGMASK	***OS-9 Technical Manual***

signal()

Specify Signal Handling

Syntax

```
#include <signal.h>
void (*signal(int sig, void (*func)(int)))(int);
```

Description

`signal()` chooses one of three ways in which receipt of the signal number `sig` is handled.

- If `func` is `SIG_DFL`, default handling for that signal occurs.
- If `func` is `SIG_IGN`, the signal is ignored.
- Otherwise, `func` is a pointer to a function to call when that signal occurs. Such a function is called a **signal handler**.

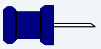
When a signal occurs, signals are masked and `signal(sig, SIG_DFL)` is executed. Then, the specified type of handling is executed. Signals are blocked for the duration of the signal handler, unless the program takes some action that unmasks the signal.

Note

A program will not receive an abort or quit signal from the keyboard (usually ^C and ^E) unless the program performs output to the terminal. This is because the operating system sends the abort/quit signals to the last process to perform I/O to the terminal. If you run the program from the shell and type ^E before the program performs I/O to the terminal, the shell receives the signal and kills the running program. If a program requires immediate control of the terminal, perform some I/O to one of the standard paths such as printing a program banner or getting the terminal options with `_gs_opt()`.

For More Information

Refer to your operating system technical manual for more information.



Note

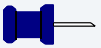
For each signal recognized by `signal()`, the default handling is program termination (with an exit status related to the exception or the signal number for non-exception related signals) and the program startup condition is `SIG_DFL`.

The default handling is reset on all signals.

Default Handling

If you specify default handling, the process terminates and the termination status is set to the signal number that occurred. This occurs if no `signal()` has been called for the signal or `signal()` was called with `SIG_DFL` as `func`.

If the program has used `intercept()`the signal handler assigned to the call is called and execution resumes at the point of interruption instead of terminating.



Note

Do not use `_os_intercept()` with `signal()`.

Ignore Handling

If a signal occurs that is to be ignored, the program resumes at the point of interruption.

If the occurring signal is exception-related, the process terminates with the appropriate vector as the exit status. Effectively, exception-related signals cannot be ignored.

Available Signals

The set of signals for the signal function varies with the operating system and the processor. A set of signals is also available regardless of the operating system.

ANSI “Core” Signals

Table 2-28 ANSI Core Signals

Signal	Description
`SIGABRT`	Abnormal termination, such as initiated by `abort()`.
`SIGFPE`	An erroneous arithmetic operation, such as zero divide or an operation resulting in overflow. This signal is generated on processor-specific conditions.
`SIGILL`	Detection of an invalid function image, such as illegal instruction. This signal is generated on processor-specific conditions.
`SIGINT`	Receipt of an interactive attention signal generated by a `<control>C` on OS-9 for 68K and OS-9.
`SIGSEGV`	An invalid access to storage. This signal is generated on processor-specific conditions.
`SIGTERM`	A termination request sent to the program.

OS-9 for 68K/OS-9 “Core” Signals

Table 2-29 OS-9 for 68K and OS-9 Core Signals

Signal	Description
SIGKILL	A death request sent to the program. This signal cannot be ignored or handled.
SIGWAKE	A wakeup request sent to the program. If a process is active when this signal arrives, it is ignored. Otherwise, the process is activated, but the signal handling function is not called.
SIGQUIT	Receipt of an interactive abort signal generated by a <control>E on OS-9 for 68K and OS-9.
SIGHUP	Modem hangup signal. This signal is sent to a process when a modem connection is lost.
SIGALRM	Not automatically sent. Programs could use this signal for an alarm call.
SIGPIPE	Not automatically sent.
SIGUSR1	Condition is user defined.
SIGUSR2	Condition is user defined.

680x0: OS-9 for 68K, OS-9

Table 2-30 OS-9 for 68K and OS-9 Core Signals for 680x0

Signal	Description
SIGADDR	Address error
SIGCHK	CHK instruction
SIGTRAPV	TRAPV instruction
SIGPRIV	Privilege violation
SIGTRACE	Trace exception
SIG1010	1010 line-A exception
SIG1111	1111 line-F exception

680x0: OS-9

Table 2-31 OS-9 Core Signals for 680x0

Signal	Description
SIGCOPRCV	Coprocessor protocol violation
SIGFMTERR	Format error
SIGUNIRQ	Uninitialized interrupt

80x86: OS-9

Table 2-32 OS-9 Core Signals for 80x86

Signal	Description
SIGGPROT	General protection
SIGSTACK	Stack exception
SIGSEGNP	Segment not present
SIGINVTSS	Invalid TSS
SIGDBLFLT	Double fault
SIGBNDCHK	Boundary check
SIGBRKPT	Breakpoint
SIGNMI	Non-maskable interrupt
SIGDBG	Debug exception

PowerPC: OS-9

Table 2-33 OS-9 Core Signals for PowerPC

Signal	Description
SIGALIGN	Alignment
SIGCHECK	Machine check
SIGINST	Instruction access
SIGPRIV	Privilege violation

Function Handling

If a signal which has an associated function occurs, the equivalent of `(*func)(sig)` is executed. `func` may terminate by executing a `return` statement or by calling `abort()`, `exit()`, or `longjmp()`. If `func` executes a `return` statement and `sig` was exception-related, the program exits with the appropriate vector for a status. Otherwise, the program resumes execution at the point it was interrupted.

Asynchronous Signals

Signals that occur by means other than `abort()` or `raise()` are called **asynchronous**. Signal handlers dealing with asynchronous signals should be careful about the following:

Calling any function in the library (except `signal()` with the occurring signal) that may be working with static storage structures.

Modifying any static storage structure other than a variable of type volatile `sig_atomic_t`.

If the `signal()` call can be honored, it returns `func` for the most recent call to `signal()` for the specified signal `sig`. Otherwise, `SIG_ERR` is returned and a positive value is stored in `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`abort()`
`exit()`
`intercept()`
`_os_intercept()`
`raise()`

sin()

Sine Function

Syntax

```
#include <math.h>
double sin(double x);
```

Description

`sin()` returns the sine of `x` as a double float. `x` is in radians.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

sinh()

Hyperbolic Sine Function

Syntax

```
#include <math.h>
double sinh(double x);
```

Description

`sinh()` returns the hyperbolic sine of `x` as a double float. `x` is in radians. A range error occurs, `ERANGE` stored in `errno`, if the magnitude of `x` is too large. In this case, the appropriately signed `HUGE_VAL` is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Error

`ERANGE`

sleep()

Suspend Execution for Specified Time

Syntax

```
#include <signal.h>
int sleep(unsigned seconds);
```

Description

`sleep()` suspends the calling process for the specified time. A sleep time of zero seconds sleeps indefinitely. `sleep()` returns the number of ticks remaining to sleep if awakened prematurely by a signal.

`seconds` specifies the length of time for the process to sleep.

Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_sleep()`	
`_os9_sleep()`	
`F$Sleep`	***OS-9 for 68K Technical Manual***
`F_SLEEP`	***OS-9 Technical Manual***

_sliblink()

Link to Subroutine Module

Syntax

```
#include <module.h>
mh_com *_sliblink(
     int          slib_num,
     char         *slib_name);
```

Description

`_sliblink()` links or loads the named module. Then, it places the subroutine module's execution entry pointer and static storage data pointer in the global arrays `_sublibs` and `_submems`, respectively. You can use `_subcall()` to make subsequent calls the subroutine library. `slib_num` specifies the subroutine number. `slib_name` is a pointer to the subroutine name string.

For More Information

For more information on creating, installing, and calling subroutine library modules, refer to the ***OS-9 Technical Manual***.

You can remove a subroutine module by passing a null pointer for the name of the module and specifying the subroutine number. A process can link to a maximum 16 subroutine libraries, numbered from 0 to 15.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

See Also

`_subcall()`
`F_SLINK` ***OS-9 Technical Manual***

Syntax

```
#include <stdio.h>
int sprintf(
     char         *str,
     const char   *format, ...);
```

Description

`sprintf()` is a standard C library function that performs formatted output to the array specified by `str`. `sprintf()` converts, formats, and prints any parameters as indicated by the control string. A null character is written at the end of the characters written; it is not counted as part of the returned sum. If copying takes place between overlapping objects, the behavior is undefined.

`str` is a pointer to an array into which to write the generated output.

`format` is a pointer to a control string.

`sprintf()` returns the number of characters written in the array, not counting the terminating null character.

For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `sprintf()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library
clib.l

See Also
printf()
fprintf()

sqrt()

Square Root Function

Syntax

```
#include <math.h>
double sqrt(double x);
```

Description

`sqrt()` returns the square root of `x`. `x` must not be negative. If the parameter is negative, `sqrt()` sets a domain error `EDOM` in `errno` and returns `HUGE_VAL`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Error

`EDOM`

srand()

Set Seed for Random Number Generator

Syntax

```
#include <stdlib.h>
void srand(unsigned int seed);
```

Description

`srand()` uses the parameter as a seed for a new sequence of pseudo-random numbers to be returned by subsequent calls to `rand()`. If `srand()` is then called with the same seed value, the sequence of pseudo-random numbers is repeated. If `rand()` is called before any calls to `srand()` have been made, the same sequence is generated as when `srand()` is first called with a seed value of one.

`seed` is used as a seed for a new sequence of pseudo-random numbers.

`srand()` does not return a value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`rand()`

srqcmem()

Allocate Colored Memory

Syntax

```
#include <memory.h>
char *srqcmem(
     int          bytecnt,
     int          memtype);
```

Description

`srqcmem()` is a direct hook to the `F$SRqCMem` (for OS-9 for 68K) or `_os_srqmem()` (for OS-9000) system call.

`bytecnt` is rounded to a system-defined block size. The size of the allocated block is stored in the global integer variable `_srqcsiz`. If `bytecnt` is `0xffffffff`, the largest contiguous block of free memory in the system is allocated.

`memtype` indicates the specific type of memory to allocate. `memory.h` contains definitions of the memory types that you may specify:

Table 2-34 srqcmem() Memory Types

Memory Type	Description
SYSRAM or MEM_SYS	System RAM memory
VIDEO1	Video memory for plane A
VIDEO2	Video memory for plane B
MEM_ANY	No memory type specified

If `memtype` is `MEM_ANY`, any available system memory may be allocated.

If successful, a pointer to the memory granted is returned. The returned pointer always begins on an even byte boundary. If the request was not granted, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Note

`srqcmem()` is identical to `_srqmem()` with the exception of the additional color parameter.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`free()`
`malloc()`
`_os_srqmem()`
`_os9_srqmem()`
`_os_srtmem()`

`F$SRqCMem`	***OS-9 for 68K Technical Manual***
`F_SRQMEM`	***OS-9 Technical Manual***

_srqmem()

System Memory Request

Syntax

```
#include <memory.h>
char *_srqmem(unsigned size);
```

Description

`_srqmem()` and the complementary function `_srtmem()` are provided to request and return system memory when tight control over memory allocation is required.

`_srqmem()` is a direct hook into the `F$SRqMem` (for OS-9 for 68K) or `F_SRQMEM` (for OS-9) system call.

`size` specifies the number of bytes to request. The specified `size` is rounded to a system-defined block size. A `size` of `0xffffffff` obtains the largest contiguous block of free memory in the system. The global unsigned variable `_srqsiz` may be examined to determine the actual size of the allocated block.

If successful, a pointer to the memory granted is returned. If the request was not granted, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

The returned pointer always begins on an even byte boundary. Care should be taken to preserve the value of the pointer if the memory is to be returned with `_srtmem()` or `_os_srtmem()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

_os_srqmem()
_os9_srqmem()
_os_srtmem()
F$SRqMem — ***OS-9 for 68K Technical Manual***
F_SRQMEM — ***OS-9 Technical Manual***

Syntax

```
#include <memory.h>
int _srtmem(
     unsigned     size,
     char         *ptr);
```

Description

`_srtmem()` is a direct hook into the `F$SRtMem` (**OS-9 for 68K**) or `_os_srtmem()` (**OS-9**) system call. It is used to return memory granted by `_srqmem()` or `_os_srqmem()`. Care should be taken to ensure that `size` and `ptr` are the same as those returned by `_srqmem()` or `_os_srqmem()`.

`size` specifies the number of bytes to return.

`ptr` is a pointer to the memory to return.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_srqmem()`
`_os9_srqmem()`
`_os_srtmem()`

`F$SRtMem`	***OS-9 for 68K Technical Manual***
`F_SRTMEM`	***OS-9 Technical Manual***

Syntax

```
#include <sg_codes.h>
int _ss_attr(
     int          path,
     int          attr);
```

Description

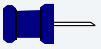
`_ss_attr()` changes a disk file's attributes. `_os_gs_fd()` determines the current attributes of a file. Only the owner of the file or the super user can change a file's attributes.

`path` is the path number of a disk file.

`attr` specifies the file attributes to set. Only the lower 16 bits of `attr` are used.

The attributes selected in `attr` are set in the file open on `path`. The `modes.h` header file defines the valid mode values.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

`_ss_attr()` is effective even if the owner or super user does not have write permission to the path. You cannot set the `directory` bit of a non-directory file or clear the `directory` bit of a directory that is not empty.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_gs_fd()`
`_os_ss_attr()`
`_os_ss_fd()`

`I$SetStt`	***OS-9 for 68K Technical Manual***
`I_SETSTAT, SS_ATTR`	***OS-9 Technical Manual***

Syntax

```
#include <stdio.h>
int sscanf(
     const char   *input,
     const char   *format, ...);
```

Description

`sscanf()` reads input from the string specified by `input`. Reaching the end of the string is equivalent to encountering end-of-file for `fscanf()`. If copying takes place between overlapping objects, the behavior is undefined.

`input` is a pointer to the input string.

`format` is a pointer to the control string.

`sscanf()` returns `EOF` if an input failure occurs before any conversion. Otherwise, `sscanf()` returns the number of input items assigned. This can be fewer than provided for or zero, in the event of an early matching failure.

For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `sscanf()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

scanf()
fscanf()
printf()

Syntax

```
#include <scf.h>
int _ss_dcoff(
     int          path,
     int          sigcode);
```

Description

`_ss_dcoff()` sends a signal to the calling process when the “data carrier detect line” associated with the device is lost.

`path` is the path number of the device.

`sigcode` is the signal code to send. Only the lower 16 bits of `sigcode` are used on OS-9 for 68K.

If `_ss_dcoff()` cannot register the signal, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ss_dcoff()`	
`I$SetStt`	***OS-9 for 68K Technical Manual***
`I_SETSTAT, SS_DCOFF`	***OS-9 Technical Manual***

_ss_dcon()

Send Data Carrier Present Signal to Process

Syntax

```
#include <scf.h>
int _ss_dcon(
     int          path,
     int          sigcode);
```

Description

`_ss_dcon()` sends a signal to the calling process when the “data carrier detect line” associated with the device is present.

`path` is the path number of the device.

`sigcode` is the signal code to send. Only the lower 16 bits of `sigcode` are used on OS-9 for 68K.

If `_ss_dcon()` cannot register the signal, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ss_dcon()`	
`I$SetStt`	***OS-9 for 68K Technical Manual***
`I_SETSTAT, SS_DCON`	***OS-9 Technical Manual***

Syntax

```
#include <scf.h>
int _ss_dsrts(int path);
```

Description

`_ss_dsrts()` disables the RTS line for an open device.

`path` is the path number of the device.

If `_ss_dsrts()` cannot disable the RTS line, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ss_dsrts()`	
`I$SetStt`	***OS-9 for 68K Technical Manual***
`I_SETSTAT, SS_DSRTS`	***OS-9 Technical Manual***

_ss_enrts()

Enable RTS Line

Syntax

```
#include <scf.h>
int _ss_enrts(int path);
```

Description

`_ss_enrts()` enables the RTS line for an open device.

`path` is the path number of the device.

If `_ss_enrts()` cannot enable the RTS line, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ss_enrts()`	
`I$SetStt`	***OS-9 for 68K Technical Manual***
`I_SETSTAT, SS_ENRTS`	***OS-9 Technical Manual***

Syntax

```
#include <rbf.h>
int _ss_lock(
     int          path,
     unsigned     locksize);
```

Description

`_ss_lock()` locks out a section of the file open on `path` from the current file position up to the number of bytes specified by `locksize`.

`path` is the path number of the file.

`locksize` specifies the number of bytes in the file to lock.

If `locksize` is zero, all locks (record-lock, EOF lock, and file lock) are removed.

If `locksize` is `0xffffffff`, the entire file is locked out regardless of the file pointer's location. This is a special type of file lock that remains in effect until released by `_ss_lock(path,0)`, a read or write of zero bytes, or the file is closed.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

For More Information

Refer to your operating system technical manual for information on RBF record locking.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ss_lock()`

`I$SetStt`	***OS-9 for 68K Technical Manual***
`I_SETSTAT, SS_LOCK`	***OS-9 Technical Manual***

Syntax

```
#include <sg_codes.h>
#include <sgstat.h>
int _ss_opt(
     int          path,
     struct sgbuf *buffer);
```

Description

`_ss_opt()` copies a buffer into the options section of the path descriptor open on `path`.

`path` specifies the path number of the path descriptor.

`buffer` is a pointer to a buffer containing the path descriptor information.

Generally, a program gets the options with `_gs_opt()`, changes the appropriate values, and updates the path options with `_ss_opt()`. The structure `sgbuf` declared in the `sgstat.h` header file provides a convenient means to access the individual option values.

If the path was invalid, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Note

It is common practice to preserve a copy of the original options so the program can restore them before exiting. The option changes take effect on the currently open path (and any paths created with `I$Dup` or `I_DUP` to the same).

Note

If you use `_ss_popt()` on OS-9, you must use the OS-9 for 68K header file. `_ss_luopt()` is provided on OS-9 solely for compatibility with OS-9. If the OS-9 for 68K header file is unavailable, change the source code to use `I_SETSTAT, SS_PATHOPT` or `I_SETSTAT, SS_LUOPT`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_gs_popt()`
`_os_ss_popt()`

`I$SetStt`	***OS-9 for 68K Technical Manual***
`I_SETSTAT`	***OS-9 Technical Manual***
`tmode`	***Using OS-9 for 68K*** or ***Using OS-9***

Syntax

```
#include <rbf.h>
#include <direct.h>
int _ss_pfd(
     int          path,
     struct fd    *buffer);
```

Description

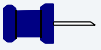
`_ss_pfd()` copies certain fields from the file descriptor information pointed to by `buffer` into the file descriptor sector of the file open on `path`. The buffer is usually obtained from `_gs_gfd()`. Only the owner ID, the modification date, and creation date fields are changed.

`path` is the path number of the file.

`buffer` is a pointer to the buffer containing the file descriptor information.

The structure `fd` declared in the `direct.h` header file provides a convenient means to access the file descriptor information.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

The buffer must be at least 32 bytes long or unanticipated data could be written into the file descriptor sector. Only the super user can change the owner ID field. You cannot change the file attributes with this call. Instead, use `_ss_attr()`.

For More Information

Refer to your operating system technical manual for information on the RBF File Manager.

Note

You must use the OS-9 for 68K `direct.h` header file. This call is provided on OS-9 solely for compatibility with OS-9 for 68K. If speed is an issue on OS-9, use `_os_ss_fd`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_gs_fd()`
`_os_ss_fd()`
`I$SetStt` — ***OS-9 for 68K Technical Manual***
`I_SETSTAT` — ***OS-9 Technical Manual***

Syntax

```
#include <sg_codes.h>
int _ss_rel(int path);
```

Description

`_ss_rel()` cancels the signal to send from a device on data ready.

`path` is the device's path number.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Note

The signal request is also cancelled when the issuing process dies or closes the path to the device. This feature exists only on SCF and PIPEMAN devices.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ss_relea()`
`_os_ss_sendsig()`
`I$SetStt` — ***OS-9 for 68K Technical Manual***
`I_SETSTAT, SS_RELEASE` — ***OS-9 Technical Manual***

_ss_rest()

Restore Device

Syntax

```
#include <sg_codes.h>
int _ss_rest(int path);
```

Description

`_ss_rest()` causes an RBF device to restore the disk head to track zero. It is usually used for disk formatting and error recovery.

`path` is the device's path number.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ss_reset()`	
`I$SetStt`	***OS-9 for 68K Technical Manual***
`I_SETSTAT, SS_RESET`	***OS-9 Technical Manual***

Syntax

```
#include <sg_codes.h>
int _ss_size(
     int          path,
     int          size);
```

Description

`_ss_size()` changes the size of an open file. This call is effective only on RBF devices. The size change is immediate. If the file size is decreased, the freed sectors are returned to the system. If the file size is increased, sectors with undefined contents are added to the file.

`path` is the file's path number.

`size` specifies the new file size in bytes.

If the path is invalid or the device is not an RBF device, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ss_size()`	
`I$SetStt`	***OS-9 for 68K Technical Manual***
`I_SETSTAT, SS_SIZE`	***OS-9 Technical Manual***

_ss_ssig()

Send Signal on Data Ready

Syntax

```
#include <sg_codes.h>
int _ss_ssig(
     int          path,
     int          sigcode);
```

Description

`_ss_ssig()` sets up a signal to be sent to the calling process when an interactive device has data ready. This feature exists only on SCF devices and pipes. When data is received on the device indicated by `path`, a signal is sent to the calling process.

`path` is the device's path number.

`sigcode` specifies the signal code to send. Only the lower 16 bits of `sigcode` are used on OS-9 for 68K.

`_ss_ssig()` must be called each time the signal is sent if it is to be used again.

The device is considered busy and returns an error if any read requests arrive before the signal is sent. Write requests to the device are allowed while in this state.

If an error occurs, `-1` is returned and the appropriate error value is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

_os_ss_relea()	
_os_ss_sendsig()	
I$SetStt	***OS-9 for 68K Technical Manual***
I_SETSTAT, SS_SENDSIG	***OS-9 Technical Manual***

Syntax

```
#include <rbf.h>
int _ss_tiks(
     int          path,
     int          tickcnt);
```

Description

If a read or write request is issued for a part of a file that is locked out by another user, RBF normally sleeps indefinitely until the conflict is removed. This feature exists only on RBF devices. `_ss_tiks()` may be used to return the error `EOS_LOCK` to the program if the conflict still exists after a specified number of ticks have elapsed.

`path` specifies the file's path number.

`tickcnt` specifies the number of ticks to wait if a record-lock conflict occurs while the file is open on `path`.

- A `tickcnt` of zero (RBF's default) causes a sleep until the record is released.
- A `tickcnt` of one returns an error if the record is not released immediately.

If an error occurs, `-1` is returned and the appropriate error value is placed in the global variable `errno`.

For More Information

Refer to your operating system technical manual for information on RBF record locking.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ss_ticks()`	
`I$SetStt`	***OS-9 for 68K Technical Manual***
`I_SETSTAT, SS_TICKS`	***OS-9 Technical Manual***

Syntax

```
#include <rbf.h>
int _ss_wtrk(
     int          path,
     int          trkno,
     int          siden,
     int          ilvf,
     char         *trkbuf,
     char         *ilvptr);
```

Description

`_ss_wtrk()` performs a write-track operation on a disk drive. It is essentially a direct hook into the driver's write-track entry point.

`path` is the path on which the device is open.

`trkno` is the track number on which to write.

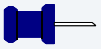
`siden` is the side of the track to write.

`ilvf` is the interleave factor.

`trkbuf` is a pointer to the track buffer image.

`ilvptr` is a pointer to the interleave table.

If an error occurs, `-1` is returned and the appropriate error value is placed in the global variable `errno`.



Note

This feature exists only on RBF devices. You can obtain additional information on actual use of this call by examining the `format` utility and/or a device driver.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ss_wtrack()`	
`I$SetStt`	***OS-9 for 68K Technical Manual***
`I_SETSTAT, SS_WTRACK`	***OS-9 Technical Manual***
`format`	***Using OS-9 for 68K*** or ***Using OS-9***

stacksiz(), _stacksiz()

Get Size of Stack Used

Syntax

```
#include <stdlib.h>
int stacksiz(void);
int _stacksiz(void);
```

Description

`stacksiz()` returns the maximum number of bytes of stack used at the time of the call if stack checking code is in effect. You can use `stacksiz()` to determine the stack size a program requires. `stacksiz()` is not available when compiling in strictly conforming ANSI mode.

Note

`stacksiz()` is historical and may be removed in a future release.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cstart.r`

stat()

Get File Status

Syntax

```
#include <UNIX/stat.h>
int stat(
     char         *path,
     struct stat  *buf);
```

Description

`stat()` obtains information about the file named by `path`. Read, write, or execute permission of the named file is not required, but all directories leading to the file must be searchable.

`buf` is a pointer to a `stat` structure into which information about the file is placed. A `stat` structure includes the following members:

```
unsigned short  st_mode;    /* attributes */
unsigned short  st_nlink;   /* nhard links */
unsigned short  st_uid;     /* user ID of owner */
unsigned short  st_gid;     /* group ID of owner */
unsigned long   st_size;    /* file size */
time_t          st_atime;   /* last access time */
time_t          st_mtime;   /* last modify time */
time_t          st_ctime;   /* last status change time */
long            st_ino;     /* undefined */
long            st_dev;     /* undefined */
long            st_rdev;    /* undefined */
```

If an error occurs, `stat()` returns `-1` and sets the global variable `errno` to indicate the error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

See Also

`fstat()`

Syntax

```
#include <strings.h>
void _strass(
     void         *to,
     void         *from,
     int          count);
```

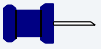
Description

`_strass()` is available to maintain source code compatibility. `count` bytes are copied from the memory pointed to by `from` to the memory pointed to by `to`, regardless of contents.

`to` is a pointer to the destination for the copy.

`from` is a pointer to the structure to copy.

`count` is the number of bytes to copy.



Note

`_strass()` can move at most 65536 bytes. No regard is given to overlapping moves.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

strcat()

String Catenation

Syntax

```
#include <string.h>
char *strcat(
     char         *orig,
     const char   *append);
```

Description

`strcat()` appends a copy of `append`, including the terminating null character, to the end of `orig`. The initial character of `append` overwrites the null character at the end of `orig`. If copying takes place between overlapping objects, the behavior is undefined.

`orig` is a pointer to the original string.

`append` is a pointer to the string to append to `orig`.

`strcat()` returns the value of `orig`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <string.h>
char *strchr(
     const char   *src,
     int          chr);
```

Description

`strchr()` locates the first occurrence of `chr`, converted to a `char`, in string `src`. The terminating null character is considered part of the string.

`src` is a pointer to the string to search.

`chr` specifies the character for which to search.

`strchr()` returns a pointer to the located character, or a null pointer if the character does not occur in the string.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

strcmp()

String Comparison

Syntax

```
#include <string.h>
int strcmp(
     const char   *string1,
     const char   *string2);
```

Description

`strcmp()` compares `string1` to `string2`.

`string1` is a pointer to a string to compare.

`string2` is a pointer to a string to compare.

`strcmp()` returns an integer that is:

- Positive, if `string1` is greater than `string2`
- Zero, if `string1` equals `string2`
- Negative, if `string1` is less than `string2`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <string.h>
int strcoll(
     const char   *string1,
     const char   *string2);
```

Description

`strcoll()` compares two strings, both interpreted as appropriate to the `LC_COLLATE` category of the current locale.

`string1` is a pointer to a string to compare.

`string2` is a pointer to a string to compare.

When both `string1` and `string2` are interpreted as appropriate to the current locale, `strcoll()` returns an integer that is:

- Positive, if `string1` is greater than `string2`.
- Zero, if `string1` equals `string2`.
- Negative, is `string1` is less than `string2`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

strcpy()

String Copy

Syntax

```
#include <string.h>
char *strcpy(
     char         *dest,
     const char   *src);
```

Description

`strcpy()` copies a string, including the terminating null character, into an array. If copying takes place between overlapping objects, the behavior is undefined.

`dest` is a pointer to the destination array.

`src` is a pointer to the string to copy into `dest`.

`strcpy()` returns the value of `dest`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <string.h>
size_t strcspn(
     const char   *src,
     const char   *delim);
```

Description

`strcspn()` returns the length of the segment of `src` up to the first occurrence of any character from `delim`.

`src` is a pointer to a source string.

`delim` is a pointer to a string of characters.

`strcspn()` returns the length of the segment.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

strerror()

Map Error Message String

Syntax

```
#include <string.h>
char *strerror(int errnum);
```

Description

`strerror()` maps an error number to an error message string. `errnum` specifies the error number. `strerror()` returns a pointer to the string. The program cannot modify the array, but a subsequent call to `strerror()` overwrites the array.

Error messages are generally in the form:

```
<major>:<minor> <detail>
```

`<major>` is the family of error.

`<minor>` is the error number within the family.

`<detail>` is the information from the system `errmsg` file, if available. The system `errmsg` file (`/dd/SYS/errmsg`, `/h0/SYS/errmsg`, or `/d0/SYS/errmsg`) contains the error messages generated by `perror()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`perror()`

strftime()

Place Formatted Time in Buffer

Syntax

```
#include <time.h>
size_t strftime(
     char         *dest,
     size_t       maxsize,
     const char   *format,
     const struct tm *timeptr);
```

Description

`strftime()` places characters into an array as controlled by `format`. The format is a multibyte character sequence, beginning and ending in its initial shift state.

`dest` is a pointer to the array into which to place the characters.

`maxsize` specifies the maximum number of characters to place in the array.

`format` is a pointer to a control string. `format` consists of zero or more conversion specifiers and ordinary multibyte characters. A conversion specifier consists of a percent `(%)` sign followed by a character that determines the behavior of the conversion specifier.

`timeptr` is a pointer to a time structure.

All ordinary multibyte characters, including the terminating null character, are copied unchanged into the array. If copying takes place between overlapping objects, the behavior is undefined. No more than `maxsize` characters are placed in the array.

Each conversion specifier is replaced by appropriate characters as described in the following list. These characters are determined by the `LC_TIME` category of the current locale and by the values contained in the structure pointed to by `timeptr`.

Table 2-35 strftime() Conversion Values

Value	Replaced By
`%a`	The locale's abbreviated weekday name.
`%A`	The locale's full weekday name.
`%b`	The locale's abbreviated month name.
`%B`	The locale's full month name.
`%c`	The locale's appropriate date and time representation.
`%d`	The day of the month as a decimal number (01 - 31).
`%H`	The hour (24-hour clock) as a decimal number (00 - 23).
`%I`	The hour (12-hour clock) as a decimal number (01 - 12).
`%j`	The day of the year as a decimal number (001 - 366).
`%m`	The month as a decimal number (01 - 12).
`%M`	The minute as a decimal number (00 - 59).
`%p`	The locale's equivalent of the AM/PM designations associated with a 12-hour clock.
`%S`	The second as a decimal number (00 - 61).
`%U`	The week number of the year (the first Sunday as the first day of week 1) as a decimal number (00 - 53).
`%w`	The weekday as a decimal number (0 - 6), where Sunday is 0.

Table 2-35 strftime() Conversion Values (continued)

Value	Replaced By
%W	The week number of the year (the first Monday as the first day of week 1) as a decimal number (00 - 53).
%x	The locale's appropriate date representation.
%X	The locale's appropriate time representation.
%y	The year without century as a decimal number (00 - 99).
%Y	The year with century as a decimal number.
%Z	The time zone name or abbreviation, or by no characters if no time zone is determinable.
%%	%.

If a conversion specifier is not one of the above, the behavior is undefined.

If the total number of resulting characters including the terminating null character is not more than `maxsize`, `strftime()` returns the number of characters placed into array `dest` not including the terminating null character. Otherwise, zero is returned and the contents of the array are indeterminate.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

strhcpy()

Copy Old OS-9 for 68K Strings

Syntax

```
#include <strings.h>
char *strhcpy(
     char         *dest,
     char         *src);
```

Description

`strhcpy()` makes a copy of `src` in `dest`. `dest` is a pointer to the destination string. `src` is a pointer to the source string to copy. `src` is assumed to have the high-order bit set on the character byte indicating the last character.

`strhcpy()` copies the bytes from `src` to `dest` until it reaches the null terminator. `strhcpy()` then removes the high-bit from the last character and appends a null byte to the output string.

`strhcpy()` is used primarily for copying file names from the directory entries on RBF disks. `strhcpy()` returns its first parameter.

Note

`strhcpy()` assumes that `dest` contains adequate space for the copy. The calling routine is responsible for verifying the space.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

strlen()

Determine String Length

Syntax

```
#include <string.h>
size_t strlen(const char *str);
```

Description

`strlen()` determines the length of `str`.

`str` is a pointer to a string.

`strlen()` returns the number of characters that precede the terminating null character.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

strncat()

String Catenation

Syntax

```
#include <string.h>
char *strncat(
     char         *orig,
     const char   *append,
     size_t       count);
```

Description

`strncat()` appends characters (a null character and characters that follow it are not appended) from `append` to the end of `orig`. The initial character of `append` overwrites the null character at the end of `orig`. A terminating null character is always appended to the result. If copying takes place between overlapping objects, the behavior is undefined.

`orig` is a pointer to the original string.

`append` is a pointer to the string to append to `orig`.

`count` specifies the maximum number of characters to append.

`strncat()` returns the value of `orig`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <string.h>
int strncmp(
     const char   *string1,
     const char   *string2,
     size_t       count);
```

Description

`strncmp()` compares not more than `count` characters from `string1` to `string2`. Characters following a null character are not compared. The strings do not need to be null terminated.

`string1` is a pointer to a string to compare.

`string2` is a pointer to a string to compare.

`count` specifies the maximum number of characters to compare.

`strncmp()` returns an integer that is:

- Positive, if `string1` is greater than `string2`.
- Zero, if `string1` equals `string2`.
- Negative, is `string1` is less than `string2`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

strncpy()

String Copy

Syntax

```
#include <string.h>
char *strncpy(char *dest, const char *src,
              size_t count);
```

Description

`strncpy()` copies characters from `src` to `dest`. Characters following a null character are not copied. If copying takes place between overlapping objects, the behavior is undefined.

`dest` is a pointer to the destination string.

`src` is a pointer to the source string to copy.

`count` specifies the maximum number of characters to copy.

If `src` is shorter than `count` characters, null characters are appended to the copy in `dest`, until `count` characters in all have been written.

`strncpy()` returns the value of `dest`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

strpbrk()

Locate First Occurrence of String

Syntax

```
#include <string.h>
char *strpbrk(
     const char   *src,
     const char   *delim);
```

Description

`strpbrk()` locates the first occurrence in `src` of any character from `delim`.

`src` is a pointer to a string in which to search for any character from `delim`.

`delim` is a pointer to a string containing characters to locate in `src`.

`strpbrk()` returns a pointer to the character or a null pointer, if no character from `delim` occurs in `src`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

strrchr()

Locate Last Occurrence of String

Syntax

```
#include <string.h>
char *strrchr(
     const char   *src,
     int          chr);
```

Description

`strrchr()` locates the last occurrence of `chr`, converted to `char`, in `src`. The terminating null character is considered part of the string.

`src` is a pointer to a string.

`chr` is the character to search for in `src`.

`strrchr()` returns a pointer to the character or a null pointer, if `chr` does not occur in the string.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

strspn()

Compute String Length

Syntax

```
#include <string.h>
char *strspn(
     const char   *src,
     const char   *pattern);
```

Description

`strspn()` returns the length of the maximum initial segment of `src` which consists entirely of characters from `pattern`.

`src` is a pointer to a string.

`pattern` is a pointer to a string.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

strstr()

Locate First Occurrence of String

Syntax

```
#include <string.h>
char *strstr(
     const char   *src,
     const char   *pattern);
```

Description

`strstr()` locates the first occurrence in `src` of the sequence of characters (excluding the terminating null character) in `pattern`.

`src` is a pointer to a string.

`pattern` is a pointer to a string containing a sequence of characters to search for in `src`.

`strstr()` returns a pointer to the located string or a null pointer, if the string is not found. If `pattern` is a pointer to a string with zero length, `strstr()` returns `src`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <stdlib.h>
double strtod(
     const char   *begin,
     char         **end);
```

Description

`strtod()` converts the initial portion of `begin` to `double` representation. `strtod()` separates the input string, attempts to convert the subject sequence to a floating point number, and then returns the result.

`begin` is a pointer to the beginning of a string.

`end` is a pointer to the pointer to the first character after the converted string.

The input string is separated into three parts:

1. An initial, possibly empty, sequence of white space characters (as specified by `isspace()`).
2. A subject sequence resembling a floating point constant. The subject sequence is the longest initial subsequence of the input string starting with the first non-white space character having the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white space character is other than a sign, a digit, or a decimal point character.
3. A final string of one or more unrecognized characters, including the terminating null character of the input string.

`strtod()` expects the subject sequence to have the following form:

- An optional plus or minus sign.
- A non-empty sequence of digits optionally containing a decimal point character.

- An optional exponent part, but no floating suffix.

If the subject sequence has the expected form, the character sequence starting with the first digit or the decimal point character (whichever occurs first) is interpreted as a floating constant. Currently, only the C locale is supported. Therefore, the decimal point character is a period. If neither an exponent part nor a decimal point character appears, a decimal point is assumed to follow the last digit in the string. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in `end`, unless `end` is a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed. `begin` is stored in `end`, unless `end` is a null pointer.

`strtod()` returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, plus or minus `HUGE_VAL` is returned, according to the sign of the value, and `ERANGE` is stored in `errno`. If the correct value would cause underflow, zero is returned and `ERANGE` is stored in `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`isspace()`
`strtol()`
`strtoul()`

Possible Error

`ERANGE`

strtok()

Break String into Tokens

Syntax

```
#include <string.h>
char *strtok(
     char         *src,
     const char   *delims);
```

Description

`strtok()` breaks `src` into a series of tokens when a sequence of calls is made to `strtok()`. A character from `delims` separates each token. When the first call is made, `src` is the first parameter. A null pointer is the first parameter for subsequent calls.

`src` is a pointer to the string to break into tokens.

`delims` is a pointer to a string containing characters to separate each token. `delims` may be different from call to call.

The first call in the sequence searches `src` for the first character that is not contained in the current `delims`. If no such character is found, then there are no tokens in `src`, and `strtok()` returns a null pointer. If a character is found, it starts the first token.

`strtok()` searches from that location for a character that is contained in the current `delims`. If the character is not found, the current token extends to the end of `src`, and subsequent searches for a token return a null pointer. If the character is found, it is overwritten by a null character to terminate the current token.

The search for the next token starts with the next character. `strtok()` saves a pointer to this location. Each subsequent call behaves in the same manner.

`strtok()` returns a pointer to the first character of a token, or a null pointer if there is no token.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

Example

```
#include <string.h>
static char str[] = “?a???b,,,#c”;
char *t;

t = strtok(str, “?”);         /* t points to the token “a” */
t = strtok(NULL, “,”);        /* t points to the token “??b” */
t = strtok(NULL, “#,”);       /* t points to the token “c” */
t = strtok(NULL, “?”);        /* t is a null pointer */
```

Syntax

```
#include <stdlib.h>
long strtol(
     const char   *begin,
     char         **end,
     int          base);
```

Description

`strtol()` converts the initial portion of `begin` to `long int` representation.

`begin` is a pointer to the beginning of the string.

`end` is a pointer to the pointer to the first character after the converted string.

`base` is the base of the number string.

`strtol()` separates the input string into three parts:

1. An initial, possibly empty, sequence of white space characters (as specified by `isspace()`).
2. A subject sequence resembling an integer represented in some radix determined by `base`. The subject sequence is the longest initial subsequence of the input string, starting with the first non-white space character of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white space character is other than a sign or a permissible letter or digit.
3. A final string of one or more unrecognized characters, including the terminating null character of the input string.

`strtol()` then attempts to convert the subject sequence to an integer and returns the result.

- If `base` is zero, the subject sequence is expected to be an integer constant, optionally preceded by a plus or minus sign. An integer suffix is not included. If the subject sequence is an integer constant, the character sequence starting with the first digit is interpreted as an integer constant.
- If `base` is between 2 and 36, the subject sequence is expected to be a sequence of letters and digits representing an integer. `base` specifies the radix, optionally preceded by a plus or minus sign. An integer suffix is not included. The letters from *a* through *z* (regardless of case) are assigned the values 10 to 35. Only letters with assigned values less than `base` are permitted. If `base` is 16, the characters 0x or 0X may optionally precede the sequence, following the sign if present.

  If the subject sequence has the expected form, it is used as the base for conversion, ascribing to each letter its value.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in `end`, unless `end` is a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed. `begin` is stored in `end`, unless `end` is a null pointer.

`strtol()` returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, `LONG_MAX` or `LONG_MIN` is returned (according to the sign of the value) and `ERANGE` is stored in `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

isupper()
strtod()
strtoul()

Possible Error

ERANGE

Syntax

```
#include <stdlib.h>
#include <limits.h>
unsigned long strtoul(
     char         *begin,
     char         **end,
     int          base);
```

Description

`strtoul()` converts the initial portion of `begin` to `unsigned long int` representation.

`begin` is a pointer to the beginning of the string.

`end` is a pointer to the pointer to the first character after the converted string.

`base` is the base of the number string.

`strtoul()` separates the input string into three parts:

1. An initial, possibly empty, sequence of white space characters (as specified by `isspace()`).
2. A subject sequence resembling an unsigned integer represented in some radix determined by `base`. The subject sequence is the longest initial subsequence of the input string, starting with the first non-white space character of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white space character is other than a sign or a permissible letter or digit.
3. A final string of one or more unrecognized characters, including the terminating null character of the input string.

`strtoul()` then attempts to convert the subject sequence to an unsigned integer and returns the result.

- If `base` is zero, `strtoul()` expects the subject sequence to be an integer constant, optionally preceded by a plus or minus sign. An integer suffix is not included.

  If the subject sequence is an integer constant, the character sequence starting with the first digit is interpreted as an integer constant.

- If `base` is between 2 and 36, the subject sequence is expected to be a sequence of letters and digits representing an integer. `base` specifies the radix, optionally preceded by a plus or minus sign. It does not include an integer suffix. The letters *a* through *z* (regardless of case) are assigned the values 10 to 35. Only letters whose assigned values are less than `base` are permitted. If `base` is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

  If the subject sequence has the expected form, it is used as the base for conversion, ascribing to each letter its value.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in `end`, unless `end` is a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed. `begin` is stored in `end`, unless `end` is a null pointer.

`strtoul()` returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, `ULONG_MAX` is returned and `ERANGE` is stored in `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

isspace()
strtod()
strtol()

Possible Error

ERANGE

Syntax

```
#include <string.h>
size_t strxfrm(
     char         *dest,
     const char   *trans,
     size_t       count);
```

Description

`strxfrm()` transforms `trans` and places the result in `dest`. You can then use `strcmp()` to compare two transformed strings. The result of the `strcmp()` on the transformed strings is the same as the result of `strcoll()` applied to the two original strings.

`dest` is a pointer to the destination string.

`trans` is a pointer to the string to transform.

`count` specifies the maximum number of characters to place in `dest`. `count` includes the terminating null character. If `count` is zero, `dest` can be a null pointer.

If copying takes place between overlapping objects, the behavior is undefined.

`strxfrm()` returns the length of the transformed string, not including the terminating null character. If the value returned is `count` or more, the contents of `dest` are indeterminate.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

See Also

strcmp()
strcoll()

Syntax

```
#include <module.h>
void _subcall(void);
```

Description

`_subcall()` calls a subroutine module previously linked by `_sliblink()`. `_subcall()` expects the first two longwords in the code area following it to contain, respectively:

- The subroutine library number.
- The function number to call.

`_subcall()` retrieves these identifiers, adjusts the program stack, and dispatches to the subroutine library entry point with the correct global static storage configuration.

For More Information

For more information, consult the ***OS-9 Technical Manual***.

Note

A process can link to a maximum of sixteen subroutine libraries, numbered from `0` to `15`.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`clib.l`

See Also

`_sliblink()`

Syntax

```
#include <time.h>
time_t sys_mktime(struct tm *tp);
```

Description

`sys_mktime()` converts the local time values in a broken-down time structure into a Universal Coordinated Time value.

`tp` is a pointer to a broken-down time structure. Broken-down time has the following format:

```
struct tm {
   int tm_sec;     /* seconds after the minute: [0,59] */
   int tm_min;     /* minutes after the hour: [0,59]   */
   int tm_hour;    /* hours after midnight:   [0,23]   */
   int tm_mday;    /* day of the month:       [1,31]    */
   int tm_mon;     /* months after January:   [0,11]   */
   int tm_year;    /* years after 1900                  */
   int tm_wday;    /* days after Sunday:      [0,6]    */
   int tm_yday;    /* days after January 1: [0,365]    */
   int tm_isdst;   /* Daylight Savings Time (DST) flag */
                   /* 0 = no, 1 = yes, -1 = unknown    */
};
```

`sys_mktime()` ignores the input values of `tm_wday` and `tm_yday`. It does not require the other fields of `tm` to be within the ranges specified in the structure. Instead, it **normalizes** the fields and sets `tm_wday`, `tm_yday`, and `tm_isdst`. This allows you to easily do temporal arithmetic without having to worry about mixed-base operations.

Note

If `(time_t)-1` returns to indicate an error, there is a one second interval that appears to be non-representable. The encoding of times for `time_t` only represents times ranging from approximately 1902 to 2038.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

`mktime()`

Syntax

```
#include <time.h>
int _sysdate(
     int          format,
     int          *time,
     int          *date,
     short        *day,
     int          *tick);
```

Description

`_sysdate()` obtains the current time, date, day of week, and clock tick from the system. Note that all the parameters except `format` are pointers to the receiving locations.

`format` specifies the format of the date and time to return. `format` can be any of the following:

- `0` = Gregorian
- `2` = Gregorian with ticks
- `1` = Julian
- `3` = Julian with ticks

`time` is a pointer to the location where `_sysdate()` stores the time value.

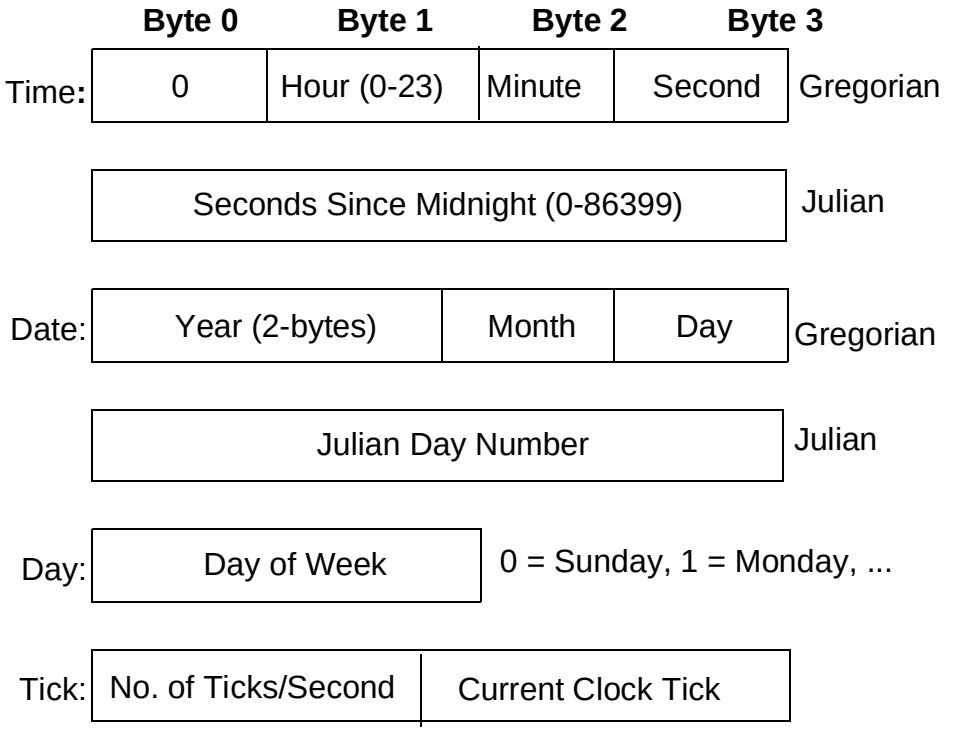
`date` is a pointer to the location where `_sysdate()` stores the date value.

`day` is a pointer to the location where `_sysdate()` stores the day of week value.

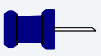
`tick` is a pointer to the location where `_sysdate()` stores the tick value.

The values are returned in the following format:

Figure 2-10 _sysdate() Return Format



If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

Be careful to pass pointers to the `date`, `time`, and `day` values. Also, be sure `day` is declared to be a `short` or the value appears as `day + 65536.`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

Example

```
main()
{
   int date,time,tick;
   short day;
   .
   .
   .
   _sysdate(0,&time,&date,&day,&tick);
   .
   :
}
```

See Also

`_os_getime()`	
`_os_julian()`	
`F$Julian`	***OS-9 for 68K Technical Manual***
`F$Time`	***OS-9 for 68K Technical Manual***
`F_TIME`	***OS-9 Technical Manual***

_sysdbg()

Call System Debugger

Syntax

```
#include <process.h>
int _sysdbg(void);
```

Description

`_sysdbg()` starts the system-level debugger, if one exists. This allows you to debug system state routines, such as drivers. The system-level debugger runs in system state and effectively stops timesharing whenever it is active. Only the super user can make this call.

Note

If the system does not have a system-level debugger, the system is reset.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_sysdbg()`
`F$SysDbg` ***OS-9 for 68K Technical Manual***
`F_SYSDBG` ***OS-9 Technical Manual***

Syntax

```
#include <stdlib.h>
int system(const char *string);
```

Description

`system()` passes a string to the shell for execution. A null pointer may be used for the string to inquire whether a command processor exists. `string` is a pointer to the string to pass to the shell. The string may be any valid shell command line.

- If `string` is a null pointer, `system()` returns non-zero only if a command processor is available.
- If `string` is not a null pointer, `system()` returns the exit status of the forked shell.

The shell used to execute the string is either the current value of the `SHELL` environment variable or `shell`, if `SHELL` is unavailable.

Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

system_addr()

Address Translation Function

Syntax

```
#include <regs.h>
void *system_addr(void *addr);
```

Description

`system_addr()` returns the system-state version of the specified address. If the passed address is already in the system-state address space or address translation is not applicable on the host processor, it is returned untouched.

`addr` is the address to be converted to its system-state address.

`system_addr()` returns the system-state version of the specified address.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

See Also

`user_addr()`

tan()

Tangent Function

Syntax

```
#include <math.h>
double tan(double x);
```

Description

`tan()` returns the tangent of `x`. `x` is presumed to be in radians.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

tanh()

Hyperbolic Tangent Function

Syntax

```
#include <math.h>
double tanh(double x);
```

Description

`tanh()` returns the hyperbolic tangent of `x`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

telldir()

Return Current Location

Syntax

```
#include <dir.h>
long telldir(DIR *dirp);
```

Description

`telldir()` returns the current location associated with the named directory stream.

`dirp` is a pointer to the directory stream.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`closedir()`
`opendir()`
`readdir()`
`rewinddir()`
`seekdir()`

tempnam()

Create Names for Temporary Files

Syntax

```
#include <UNIX/os9def.h>
char *tempnam(
     char         *dir,
     char         *pfx);
```

Description

`tempnam()` generates names that can safely be used for temporary files. `tempnam()` allows the user to control the choice of a directory. The argument `dir` points to the name of the directory in which the file is to be created. If `dir` is `NULL` or points to a string which is not a name for an appropriate directory, “.” is used. This entire sequence can be preempted by setting the environment variable `TMPDIR`, whose value is the name of the desired temporary-file directory.

Many applications prefer their temporary files to have certain favorite initial letter sequences in their names. Use the `pfx` argument for this. This argument may be NULL or point to a string to be used as the first few characters of the temporary-file name.

`tempnam()` uses `malloc()` to get space for the constructed file name, and returns a pointer to this area. Thus any pointer value returned from `tempnam()` may serve as an argument to `free()`. If `tempnam()` cannot return the expected result for any reason, that is `malloc()` failed, or none of the above mentioned attempts to find an appropriate directory was successful, a NULL pointer is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

tgetent()

Get Termcap Entries

Syntax

```
#include <termcap.h>
extern char *BC;
extern char *UP;
int tgetent(
     char         *bufptr,
     char         *name);
```

Description

`tgetent()` extracts the entry for the specified terminal type. The entry is placed in a buffer. The buffer size must be at least 1,024 characters and must remain intact for all subsequent calls to `tgetnum()`, `tgetflag()`, and `tgetstr()`.

`bufptr` is a pointer to the buffer in which to place the entry.

`name` is a pointer to the name of the terminal type.

If the `termcap` file cannot be opened, `-1` is returned. If the terminal name cannot be found, `0` is returned. If the terminal name is found, `1` is returned and the data is placed in the buffer.

By defining the shell environment variables `TERMCAP` and `TERM`, you can modify the behavior of `tgetent()`. If `TERMCAP` is found in the environment and its value string begins with a slash (`/`), that string is used as the path to the file to use instead of the default `termcap` file. If the value string does not begin with a slash and `name` is the same as the environment string for `TERM`, `tgetent()` uses this file to search for the terminal entry. This mode is useful for testing or for custom `termcap` definitions. If the `TERMCAP` value string does not begin with a slash, the string is used as the `termcap` string instead of reading a file.

Note

`tgetent()` must be called before any of the other `termcap` library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`termlib.l`

See Also

`tgetflag()`
`tgetnum()`
`tgetstr()`

tgetflag()

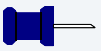
Check Terminal Capability Presence

Syntax

```
#include <termcap.h>
extern char *BC;
extern char *UP;
int tgetflag(char *id);
```

Description

`tgetflag()` returns `1` if the capability ID was specified for the terminal type and `0` if not specified for the terminal type.



Note

`tgetent()` must be called before any of the other `termcap` library functions.

`id` is a pointer to the capability ID.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`termlib.l`

tgetnum()

Get Terminal Capability ID

Syntax

```
#include <termcap.h>
extern char *BC;
extern char *UP;
int tgetnum(char *id);
```

Description

`tgetnum()` returns the numeric value given for the capability ID.

Note

`tgetent()` must be called before any of the other `termcap` library functions.

`id` is a pointer to the capability ID.

If `id` was not specified for the terminal type, `-1` is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

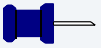
`termlib.l`

Syntax

```
#include <termcap.h>
extern char *BC;
extern char *UP;
char *tgetstr(
     char         *id,
     char         **strptr);
```

Description

`tgetstr()` places the string given for the capability ID into a buffer.



Note

`tgetent()` must be called before any of the other `termcap` library functions.

`id` is a pointer to the capability ID.

`strptr` is a pointer to the pointer to the buffer for the capability string. `strptr` is advanced past the last character of the returned data.

`tgetstr()` returns null or `0` upon error.

Cursor motion and padding information are interpreted when the string is actually output with `tgoto()` and `tputs()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

termlib.l

See Also

tgoto()
tputs()

tgoto()

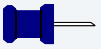
Get Cursor Movement Capability

Syntax

```
#include <termcap.h>
extern char *BC;
extern char *UP;
char *tgoto(
     char         *motion_string,
     int          column,
     int          line);
```

Description

`tgoto()` returns a string suitable for positioning the cursor on the terminal.



Note

`tgetent()` must be called before any of the other `termcap` library functions.

`motion_string` is a pointer to the string given by the `cm` capability.

`column` specifies the column destination for the cursor.

`line` specifies the line destination for the cursor.

The `extern` variables `UP` and `BC` (set to the up and `bc` capability strings, respectively) are used to avoid placing a null or newline character into the output string.

`tgoto()` returns a pointer to the translated motion string if the motion string was successfully created. Otherwise, the string `"OOPS"` is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`termlib.l`

time()

Get Calendar Time

Syntax

```
#include <time.h>
time_t time(time_t *timer);
```

Description

`time()` determines the current calendar time. The encoding of the value is unspecified.

`timer` is a returned value. It is a pointer to the calendar time.

`time()` returns the implementation's best approximation to the current calendar time. The value `(time_t)-1` is returned if the calendar time is unavailable. If `timer` is not a null pointer, the return value is also assigned to the object to which it points.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`getenv()`
`_os_getime()`
`_os9_getime()`

times()

Get Process Times

Syntax

```
#include <UNIX/times.h>
int times(struct tms *buffer);
```

Description

`times()` returns time-accounting information for the current process and for the terminated child processes of the current process. All times are in 1/60 seconds.

`buffer` points to the following structure:

```
struct tms {
     clock_t   tms_utime;   /* user time */
     clock_t   tms_stime;   /* system time */
     clock_t   tms_cutime;  /* user time, children */
     clock_t   tms_cstime;  /* system time, children */
};
```

This information comes from the calling process.

`tms_utime` is the CPU time used while executing instructions in the user space of the calling process.

`tms_stime` is the CPU time used by the system on behalf of the calling process.

`tms_cutime` and `tms_cutimes` are not implemented and set to zero by `times`().

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

tmpfile()

Create Temporary Binary File

Syntax

```
#include <stdio.h>
FILE *tmpfile(void);
```

Description

`tmpfile()` creates a temporary binary file that is automatically removed when it is closed or at program termination. If the program terminates abnormally, temporary files are not removed. The file is opened for update with `"wb+"` mode.

`tmpfile()` returns a pointer to the stream of the file that it created. If the file cannot be created, `tmpfile()` returns a null pointer.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fopen()`

tmpnam()

Generate Unique Valid Filename

Syntax

```
#include <stdio.h>
char *tmpnam(char *filename);
```

Description

`tmpnam()` generates a string. The string is a valid file name and is not the same as the name of an existing file. `tmpnam()` generates a different string each time it is called up to `TMP_MAX` times. If it is called more than `TMP_MAX` times, it returns a null pointer.

`filename` is a pointer to the generated file name.

- If `filename` is a null pointer, `tmpnam()` leaves its result in an internal static object and returns a pointer to that object. Subsequent calls to `tmpnam()` modify the same object.
- If `filename` is not a null pointer, it is assumed to point to an array of at least `L_tmpnam` chars. `tmpnam()` writes its result in that array and returns the parameter as its value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <ctype.h>
int toascii(int c);
int _toascii(int c);
```

Description

`_toascii()` returns a character with all bits that are not part of a standard ASCII character turned off. It is used for compatibility with other systems. Only `_toascii()` is available in strictly conforming ANSI mode.

c is the returned character.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`clib.l`

See Also

`getc(), getchar()`

_tolower()

Convert Character to Lowercase

Syntax

```
#include <ctype.h>
int _tolower(int c);
```

Description

`_tolower()` is a macro that changes the uppercase parameter to lowercase. The parameter **must** be uppercase or the results are unpredictable and useless. Use `tolower()` if the parameter is not guaranteed to be uppercase. c is the character to convert. Only the lower 8 bits of c are used.

`_tolower()` is hard-coded to the ASCII character set. Changing the locale does not cause `_tolower()` to recognize a non-ASCII character. You must use `tolower()` if you are not using ASCII characters.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`ctype.h`.

See Also

`tolower()`
`toupper()`
`_isascii(), isascii()`

tolower()

Convert Character to Lowercase

Syntax

```
#include <ctype.h>
int tolower(int c);
```

Description

`tolower()` converts an uppercase letter to the corresponding lowercase letter.

`c` is the character to convert.

If the parameter `c` is such that `islower(c)` returns a non-zero value (is uppercase), `tolower()` returns the corresponding character. Otherwise, `c` is returned unchanged.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`islower()`

_toupper()

Convert Character to Uppercase

Syntax

```
#include <ctype.h>
int _toupper(int c);
```

Description

`_toupper()` is a macro that changes the lowercase parameter to uppercase. The parameter **must** be lowercase or the results are unpredictable and useless. Use `toupper()` if the parameter is not guaranteed to be lowercase.

c is the character to convert. Only the lower 8 bits of c are used.

`_toupper()` is hard-coded to the ASCII character set. Changing the locale does not cause `_toupper()` to recognize a non-ASCII character. You must use `toupper()` if you are not using ASCII characters.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`ctype.h`

See Also

`tolower()`
`toupper()`
`_isascii(), isascii()`

toupper()

Convert Character to Uppercase

Syntax

```
#include <ctype.h>
int toupper(int c);
```

Description

`toupper()` converts a lowercase letter to the corresponding uppercase letter.

`c` is the character to convert.

If the parameter `c` is such that `isupper(c)` returns a non-zero value (is uppercase), `toupper()` returns the corresponding character. Otherwise, `c` is returned unchanged.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`isupper()`

tputs()

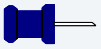
Output Capability String

Syntax

```
#include <termcap.h>
extern char *BC;
extern char *UP;
void tputs(
     char         *str,
     int          lines_affected,
     int          (*outfunc)(void));
```

Description

`tputs()` is used to output the capability strings to the terminal. `tputs()` decodes leading padding information from the `str` string.



Note

`tgetent()` must be called before any of the other `termcap` library functions.

`str` is a pointer to the string.

`lines_affected` is the number of lines the operation affects. This should be set to `1` if not applicable.

`outfunc` is a pointer to a function called by `tputs()` to output each successive character of the `str` string.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library
termlib.l

See Also
_gs_opt()

tsleep()

Sleep for Specified Interval

Syntax

```
#include <signal.h>
int tsleep(unsigned svalue);
```

Description

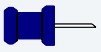
`tsleep()` deactivates the calling process for a specified interval.

`svalue` specifies the interval:

- If `svalue` is `0`, the process sleeps indefinitely.
- If `svalue` is `1`, the process gives up the current time slice.

`For values greater than one, svalue` is considered a tick count to sleep, if the high bit is clear. If the high bit of `svalue` is set, the remaining 31 bits are considered the number of 256ths of a second to sleep.

If the sleeping process is awakened prematurely by a signal, `tsleep()` returns the number of ticks remaining to sleep.



Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

_os_sleep()	
_os9_sleep()	
F$Sleep	***OS-9 for 68K Technical Manual***
F_SLEEP	***OS-9 Technical Manual***

ungetc()

Unget Character

Syntax

```
#include <stdio.h>
int ungetc(
     int          c,
     FILE         *stream);
```

Description

`ungetc()` pushes a character, converted to an `unsigned char`, back onto the input stream. The pushed-back character is returned by subsequent reads on that stream in the reverse order of their pushing. A successful intervening call with the stream pointed to by `stream` to a file positioning function (`fseek()`, `fsetpos()`, or `rewind()`) discards any pushed-back characters for the stream. The external storage corresponding to the stream is unchanged.

`c` is the character to push back. If `c` is `EOF`, the operation fails and the input stream is unchanged.

`stream` is a pointer to the C I/O `FILE` structure.

One character of pushback is guaranteed. If `ungetc()` is called too many times on the same stream without an intervening read or file positioning operation on that stream, the operation may fail.

A successful call to `ungetc()` clears the end-of-file indicator for the stream. The value of the file position indicator for the stream after reading or discarding all pushed-back characters is the same as it was before the characters were pushed back. For a text stream, the value of the file position indicator after a successful call to `ungetc()` is unspecified until all pushed-back characters are read or discarded. For a binary stream, the file position indicator is decremented by each successful call to `ungetc()`. If the value was zero before a call, it is indeterminate after the call.

`ungetc()` returns the character pushed back after conversion. If the conversion fails, `EOF` is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`getc(), getchar()`
`fopen()`
`fseek()`
`fsetpos()`
`rewind()`
`setbuf()`

unlink()

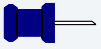
Unlink (Delete) File

Syntax

```
#include <modes.h>
int unlink(char *name);
```

Description

`unlink()` decrements the link count of a file. When the link count reaches zero (0), the space occupied by the file on the disk is freed and the file no longer exists. `name` is a pointer to the name of the file. If successful, `unlink()` returns `0`. Otherwise, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

OS-9 for 68K does not yet support multiple links to a file. `unlink()` always causes the file to be removed from the disk. Attempting to delete an open file by the calling (or another) process results in an `EOS_SHARE` error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_delete()`	
`I$Delete`	***OS-9 for 68K Technical Manual***
`I_DELETE`	***OS-9 Technical Manual***

unlinkx()

Unlink (Delete) File

Syntax

```
#include <modes.h>
int unlinkx(
     char         *name,
     int          mode);
```

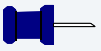
Description

`unlinkx()` decrements the link count of a file.

`name` is a pointer to the name of the file.

`mode` specifies the file's access mode. If the execution bit (`FAM_EXEC` or `S_IEXEC`) in the `mode` is set, the path is assumed to be based in the current execution directory. The header file `modes.h` defines the legal `mode` values. When the link count reaches zero, the space occupied by the file on the disk is freed and the file no longer exists. Only the lower 16 bits of `mode` are used.

If successful, `unlinkx()` returns `0`. Otherwise, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

OS-9 for 68K does not support multiple links to a file. `unlinkx()` always causes the file to be removed from the disk. Attempting to delete an open file by the calling (or another) process results in an `EOS_SHARE` error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

_os_delete()
I$Delete — *OS-9 for 68K Technical Manual*
I_DELETE — *OS-9 Technical Manual*

Syntax

```
#include <regs.h>
void *user_addr(void *addr);
```

Description

`user_addr()` returns the user-state version of the specified address. If the passed address is already in the user-state address space or address translation is not applicable on the host processor, it is returned untouched.

`addr` is the address to be converted to its user-state address.

`user_addr()` returns the user-state version of the specified address.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

See Also

`system_addr()`

va_arg()

Get Parameter in Variable Parameter List

Syntax

```
#include <stdarg.h>
type va_arg(va_list ap, type);
```

Description

`va_arg()` is a macro associated with the `stdarg.h` header file. `stdarg.h` provides a method for stepping through a list of function parameters whose length and type are unknown.

`ap` is an object representing the next unnamed parameter.

`type` is a type name.

`va_arg()` expands into an expression. The expression returns a value that has the type and value of the next unnamed parameter. It also modifies `ap` such that the next call to `va_arg()` returns the next parameter. `ap` must have been previously initialized using `va_start()`.

If there is no actual next parameter or if `type` is not compatible with the type of the actual next parameter (as promoted according to the default parameter promotions), the behavior is undefined.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`va_start()`

va_end()

End References to Current Variable Parameter List

Syntax

```
#include <stdarg.h>
void va_end(va_list ap);
```

Description

`va_end()` is a macro associated with the `stdarg.h` header file. `stdarg.h` provides a method for stepping through a list of function parameters whose length and type are unknown.

`ap` is an object representing the next unnamed parameter.

`va_end()` expands into an expression. The expression facilitates a normal return from the function whose variable parameter list was referred to by the expansion of the `va_start()` call that initialized `ap`.

`va_end()` may modify `ap` so that you can no longer use it without an intervening call to `va_start()`. If there is no corresponding call to `va_start()` or if `va_end()` is not called before the return, the behavior is undefined. `va_end()` returns no value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

`va_start()`

va_start()

Initialize Variable Parameter List

Syntax

```
#include <stdarg.h>
void va_start(va_list ap, parmN);
```

Description

`va_start()` is a macro associated with the `stdarg.h` header file. `stdarg.h` provides a method for stepping through a list of function parameters whose length and type are unknown.

`ap` is an object representing the next unnamed parameter.

`parmN` identifies the right most parameter in the variable parameter list in the function definition (the parameter just before the `, ...`). If `parmN` is declared with one of the following types, the behavior is undefined:

- The register storage class
- A function type
- An array type
- A type that is not compatible with the type that results after applying the default parameter promotions. For example, `char`, `short`, and `float`.

`va_start()` returns no value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <stdarg.h>
#include <stdio.h>
int vfprintf(
     FILE         *stream,
     const char   *format,
     va_list      arg);
```

Description

`vfprintf()` performs formatted output to a stream. It converts, formats, and prints any parameters as indicated by the control string. `vfprintf()` is similar to `fprintf()`. However, `arg` replaces the variable parameter list.

`stream` is a pointer the C I/O `FILE` structure.

`format` is a pointer to a control string.

`arg` is initialized by `va_start()` (and possibly subsequent `va_arg()` calls).

`vfprintf()` does not call `va_end()`. `vfprintf()` returns the number of characters transmitted. If an output error occurs, a negative value is returned.

For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `vfprintf()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

Example

The following shows the use of `vfprintf()` in a general error-reporting routine.

```
#include <stdarg.h>
#include <stdio.h>
void error(char *function_name, char *format, ...)
{
     va_list args;
     va_start(args, format);   /* print out name of function */
                               /* causing error */
     fprintf(stderr, “ERROR in %s “, function_name);
                               /* print out remainder of message */
     vfprintf(stderr, format, args);
     va_end(args);
}
```

See Also

`fprintf()`
`va_arg()`
`va_end()`
`va_start()`

Syntax

```
#include <stdarg.h>
#include <stdio.h>
int vprintf(
     const char   *format,
     va_list      arg);
```

Description

`vprintf()` performs formatted output to `stdout`. It converts, formats, and prints any parameters as indicated by the control string. `vprintf()` is similar to `printf()`. However, `arg` replaces the variable parameter list.

`format` is a pointer to a control string.

`arg` is initialized by `va_start()` (and possibly subsequent `va_arg()` calls).

`vprintf()` does not call `va_end()`.

`vprintf()` returns the number of characters transmitted. If an output error occurs, a negative value is returned.

For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `vprintf()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fprintf()`
`printf()`
`va_arg()`
`va_end()`
`va_start()`

vsprintf()

Print to String

Syntax

```
#include <stdarg.h>
#include <stdio.h>
int vsprintf(
     char         *str,
     const char   *format,
     va_list      arg);
```

Description

`vsprintf()` performs formatted output to a string. It converts, formats, and prints any parameters as indicated by the control string. `vsprintf()` is similar to `sprintf()`. However, `arg` replaces the variable parameter list.

`str` is a pointer to a string.

`format` is a pointer to a control string.

`arg` is initialized by `va_start()` (and possibly subsequent `va_arg()` calls).

`vsprintf()` does not call `va_end()`. If copying takes place between overlapping objects, the behavior is undefined.

`vsprintf()` returns the number of characters written in the array, not counting the terminating null character.

For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `vsprintf()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fprintf()`
`sprintf()`
`va_arg()`
`va_end()`
`va_start()`

Syntax

```
#include <process.h>
int wait(unsigned int *status);
int wait(0);
```

Description

`wait()` suspends the current process until a child process has terminated.

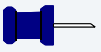
`status` is a pointer to a storage location where the exit status of the terminating child process is stored.

`wait()` returns the process ID of the terminating child process and places the exit status of that process in the `unsigned int` pointed to by `status`. If `status` is passed as zero, no child status is available to the caller.

The lower 16 bits of the status value contains the parameter of the `exit()` or `_exit()` call as executed by the child process, or the signal number if it was interrupted with a signal. A normally terminating C program with no explicit call to `exit()` in `main()` returns a status equal to the returned value of `main()`.

`wait()` returns:

- `-1` if there is no child process for which to wait.
- `0` if a signal was received before a child process terminated.



Note

The status codes used in the operating system may not be compatible with other operating systems. A wait must be executed for each child process created.

Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`exit()`
`_exit()`
`_os_wait()`
`_os_fork()`

`F$Wait`	***OS-9 for 68K Technical Manual***
`F_WAIT`	***OS-9 Technical Manual***

Syntax

```
#include <sys/wait.h>
pid_t waitpid(pid_t pid, int *stat_loc, int options);
```

Description

`waitpid()` allows a thread to wait for a process and get its exit status. The `pid` and `options` arguments determine how the wait is done. For values of `pid` less than or equal to 0, the next available child process' status is returned. For values of `pid` greater than 1 (1 is an invalid process ID) the exit status of the specified process is returned. The `options` parameter can either be 0 to indicate a blocking wait or `WNOHANG` to indicate a non-blocking wait.

If or when a child's exit status is available `waitpid()` returns the process ID and, if `stat_loc` is non-NULL, writes the exit status at `stat_loc`. If a blocking `waitpid()` is interrupted by a signal it sets `errno` to `EINTR` and returns -1. If `options` is `WNOHANG` and the calling process has children, but none of the specified children has available exit status, 0 is returned. Otherwise `errno` is set to the error number and -1 is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

Possible Errors

`ECHLD`	The calling process has no children or not the specific child requested.

EINTR	A blocking `waitpid()` was interrupted by a signal.
EINVAL	The value of `options` is invalid.

Example

```
while (waitpid(child, &status, 0) == -1 && errno == EINTR)
   ;
```

See Also

`wait()`
`pthread_join()`
`pthread_exit()`

Syntax

```
#include <stdlib.h>
size_t wcstombs(
     char         *str,
     const wchar_t *pwcs,
     size_t       maxnum);
```

Description

`wcstombs()` converts a sequence of codes that correspond to multibyte characters from an array into a sequence of multibyte characters that begins in the initial shift state. `wcstombs()` stores these multibyte characters, stopping if a multibyte character would exceed the limit of `maxnum` total bytes or if a null character is stored. Each code is converted as if by a call to `wctomb()`, except that the shift state of `wctomb()` is not affected.

`str` is a pointer to the string in which to store the multibyte characters. `pwcs` is a pointer to the array from which to read. `maxnum` specifies the maximum number of bytes to convert.

No more than `maxnum` bytes are modified in `str`. If copying takes place between overlapping objects, the behavior is undefined.

If a code is encountered that does not correspond to a valid multibyte character, `wcstombs()` returns `(size_t)-1`. Otherwise, `wcstombs()` returns the number of bytes modified, not including a terminating null character, if any.

`wcstombs()` is supported for both the `C` and `JAPAN` locales.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

See Also

wctomb()

Syntax

```
#include <stdlib.h>
int wctomb(
     char         *str,
     wchar_t      wchar);
```

Description

`wctomb()` determines the number of bytes needed to represent the multibyte character corresponding to the code whose value is `wchar`. This includes any change in shift state. `wctomb()` stores the multibyte character representation in `str`, if `str` is not a null pointer. At most `MB_CUR_MAX` characters are stored.

`str` is a pointer to the string in which to store the multibyte character representation.

`wchar` is the multibyte character to be represented. If `wchar` is zero, `wctomb()` is left in the initial shift state.

If `str` is a null pointer, `wctomb()` returns the following:

- A non-zero value if multibyte character encodings have state-dependent encodings.
- Zero if multibyte character encodings do not have state-dependent encodings.

If `str` is not a null pointer, `wctomb()` returns the following:

- `-1` if the value of `wchar` does not correspond to a valid multibyte character
- The number of bytes contained in the multibyte character corresponding to the value of `wchar`.

The returned value is never greater than `MB_CUR_MAX`.

`wctomb()` is supported for both the `C` and `JAPAN` locales.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <modes.h>
int write(
     int          path,
     char         *buffer,
     unsigned     count);
```

Description

`write()` writes bytes to a path.

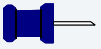
`path` is the path number. The path number is an integer which specifies one of the standard path numbers (`0`, `1`, or `2`) or a path number returned from a successful call to `open()`, `creat()`, `create()`, or `dup()`.

`buffer` is a pointer to space with at least `count` bytes of memory from which `write()` obtains the data to write to the path.

It is guaranteed that at most `count` bytes are written. If fewer bytes are written, an error has occurred.

`write()` essentially performs a **raw** `write`; that is, the `write` is performed without translating characters. The characters are passed to the file (or device) as transmitted by the program.

`write()` returns the number of bytes actually written. If an error occurred, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

`write()` performs a **raw** `write` to the terminal; no linefeeds after returns are transmitted.

Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`creat()`
`create()`
`dup()`
`open()`
`_os_write()`
`writeln()`

`I$Write`	***OS-9 for 68K Technical Manual***
`I_WRITE`	***OS-9 Technical Manual***

Syntax

```
#include <modes.h>
int writeln(
     int          path,
     char         *buffer,
     unsigned     count);
```

Description

`writeln()` writes bytes to a path.

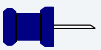
`path` is the path number. The path number is an integer which specifies one of the standard path numbers (`0`, `1`, or `2`) or a path number returned from a successful call to `open()`, `creat()`, `create()`, or `dup()`.

`buffer` is a pointer to space with at least `count` bytes of memory from which `writeln()` obtains the data to write to the path.

It is guaranteed that at most `count` bytes are written.

`writeln()` causes output-filtering to take place such as outputting a linefeed after a carriage return or handling the page pause facility. `writeln()` writes, at most, one line of data. A carriage return indicates the end of a line. `writeln()` is the preferred call for writing to the terminal.

`writeln()` returns the number of bytes actually written. If an error occurred, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

`writeln()` stops when the carriage return is written, even if the byte count has not been exhausted.

Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`creat()`
`create()`
`dup()`
`open()`
`_os_writeln()`
`write()`

`I$WritLn`	***OS-9 for 68K Technical Manual***
`I_WRITELN`	***OS-9 Technical Manual***

Syntax

```
#include <UNIX/os9def.h>;
int writev(
     unsigned int fd,
     struct iovec *iov,
     unsigned int iovcnt);
```

Description

`writev()` performs the same action as `write`(), but gathers the output data from the `iovcnt` buffers specified by the members of the `iov` array: `iov[0], iov[1], ..., iov[iovcnt-1]`. If `iovcnt` is zero, any `iov_len` member of a passed `iovec` structure is less than zero, or all the `iov_len` members of the passed `iovec` structures are zero, `writev()` returns `-1` and sets the global variable `errno`.

For `writev()`, the `iovec` structure is defined as follows.

```
struct iovec {
     caddr_t      iov_base;
     int iov_len;
};
```

Each `iovec` entry specifies the base address and length of an area in memory from which data should be written. `writev`() always writes a complete area before proceeding to the next.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

Appendix A: Prototyping sys_clib.l Functions

Before this topic is discussed in depth, it is important to note that the primary purpose of `sys_clib.l` is to provide a library for compatibility with the Microware K&R C compiler. This allows people who wrote code for that compiler to use the Ultra C/C++ ANSI C compiler without altering their original code.

Note

It is strongly recommended that when writing new code, a programmer not use the `sys_clib` library functions.

Table 1-37 provides a table to look up equivalent functions for the `sys_clib` functions.

Function Declarations and K&R Compiler vs. Prototypes and ANSI C

In the old days when K&R C ruled the earth, the concept of a function protoype did not exist. The closest equivalent concept was a "function declaration". The purpose of the function declaration was to "pre-define" what the return type of a function was so that some degree of type checking could be done on its use. An example of a function declaration is:

```
struct direct *readdir();
```

This shows that the function "`readdir`" returns a pointer to a structure of the type "`direct`" (found in `dir.h`). NOTE: There is nothing in the declaration which mentions what type, or even how many, arguments the readdir function takes. As a result, no type checking was ever done on these functions parameters. The "standard promotion" of arguments was always assumed to have occurred.

The "standard promotion" rules are quite simple, all types being passed in that are "narrower" than an int type, are "widened" to an int type, and all arguments that are of floating point type are "widened" to a double precision. As a natural consequence, all functions which are compiled in K&R mode (for instance, all the `sys_clib.l` functions) assume that the calling function has already promoted the function arguments to their widened forms.

ANSI C prototypes are somewhat different. An example of a prototype is:

```
int os9fork(char *, int, char *, short, short, int short);
```

This shows that the function "`os9fork`" returns an `int` and accepts the parameters specified. Of particular interest is the three "short" parameters. In ANSI mode, the compiler assumes that the prototype was visible to the calling program, and as a result, the values were NOT widedned to `int`. If the prototype was not visible to the calling function, the values would have been widened. This could have potentially caused a problem (especially if the parameters were passed on the

stack and NOT in registers). As a result, the Ultra C compiler has an option (`-cw`) which warns the user if a function is being called without a prototype for that function being visible.

"ANSI"fying Code and Function Prototypes

It has been a common practice as of late to take code and switch it over to ANSI-compliant code. This often entails prototyping many of the functions which used to be compiled with K&R C compilers. If this were done strictly as in the prototype from the "`os9fork`" function mentioned above, and the `os9fork` function was not recompiled with an ANSI compiler, things would not work since the function itself would be expecting the short arguments to be widened to ints and the calling program, after having seen the prototype, would send shorts.

There is really only one way to handle this in a manner that causes everything to still work and that is to prototype the functions to their widened types, as follows:

```
int os9fork(char *, int, char *, int, int, int, int);
```

Unfortunately, this has the undesired side effect of implying that the fourth, fifth and seventh arguments are valid integers, when indeed only their lower 16 bits are valid. As a result, Microware has decided to optionally prototype these functions in system header files.

`sys_clib.l` functions affected by this include:

`os9exec()`
`os9fork(), os9forkc()`
`os9kexec()`

Index

Symbols

_aton() 37
_atou() 38
_atou()alpha
to unsigned conversion 142
_cleareof() 46, 104, 168
_cmpnam() 65, 104, 175
_cpymem() 62, 104, 179
_dtoa() 38
_environ 101
_errmsg() 65, 103, 201
_ERROR 140
_ev_creat() 52, 103, 203
_ev_delete() 52, 103, 205
_ev_info() 52, 103, 206
_ev_link() 53, 103, 208
_ev_pulse() 53, 103, 209
_ev_read() 53, 103, 211
_ev_set() 53, 104, 213
_ev_setr() 53, 104, 215
_ev_signal() 53, 104, 217
_ev_unlink() 53, 104, 219
_ev_wait() 53, 104, 221
_ev_waitr() 53, 104, 223
_exit() 35, 104, 232
_fileno() 47, 249
_freemem() 62, 272
_freemin() 274
get<name>() 292
_get_module_dir() 68, 105, 313
_get_process_desc() 105, 320
_get_process_table() 76, 105, 322
_getsys() 73, 105, 336
_gregorian() 87, 106, 342

_gs_devn() 39, 106, 344
_gs_eof() 39, 106, 346
_gs_gfd() 39, 106, 348
_gs_opt() 39, 106, 350
_gs_pos() 39, 106, 352
_gs_rdy() 39, 106, 353
_gs_size() 39, 106, 355
_INTEGER 140
_IOFBF 945
_IOLBF 945
_IONBF 945
_isascii() 36, 380
_isjis_kana() 36, 387, 389, 393
_issjis1() 37, 401
_issjis2() 37, 403
_julian() 87, 107, 411
_lcalloc() 62, 416
_lfree() 62, 420
_lmalloc() 62, 423
_lrealloc() 62, 434
_mallocmin() 63, 445
_mkdata_module() 68, 105, 457
_os_acqlk() 79, 502
_os_alarm_atime() 54, 506
_os_alarm_cycle() 54, 509
_os_alarm_delete() 54, 510
_os_alarm_reset() 54, 511
_os_alarm_set() 54, 512
_os_alias() 48, 513
_os_alltsk() 63, 516
_os_alocproc() 76, 517
_os_altmdir() 69, 518
_os_aproc() 76, 519
_os_attach() 48, 521
_os_cache() 65, 523
_os_caqlk() 79, 525
_os_chain() 76, 526
_os_chainm() 69, 528
_os_chdir() 48, 530
_os_chkmem() 532
_os_chmdir() 69, 534

_os_close() 48, 535
_os_clrsigs() 54, 536
_os_cmdperm() 69, 537
_os_cmpnam() 65, 538
_os_config() 540
    configure 540
_os_cpy_ioproc() 48, 541
_os_cpymem() 63, 542
_os_crc() 69, 543
_os_create() 48, 545
_os_crlk() 79, 549
_os_datmod() 69, 550
_os_ddlk() 79, 552
_os_delete() 48, 555
_os_dellk() 79, 557
_os_delmdir() 69, 558
_os_deltsk() 63, 559
_os_detach() 49, 560
_os_dexec() 66, 561
_os_dexit() 66, 564
_os_dfork() 77, 565
_os_dforkm() 77, 569
_os_dup() 49, 571
_os_ev_allclr() 54, 573
_os_ev_allset() 54, 575
_os_ev_anyclr() 54, 576
_os_ev_anyset() 54, 577
_os_ev_change() 54, 578
_os_ev_creat() 54, 579
_os_ev_delete() 55, 581
_os_ev_info() 55, 582
_os_ev_link() 55, 585
_os_ev_pulse() 55, 586
_os_ev_read() 55, 588
_os_ev_set() 55, 589
_os_ev_setand() 55, 591
_os_ev_setor() 55, 593
_os_ev_setr() 55, 595
_os_ev_setxor() 55, 597
_os_ev_signal() 55, 599
_os_ev_tstset() 56, 601

_os_ev_unlink() 56, 603
_os_ev_wait() 56, 604
_os_ev_waitr() 56, 607
_os_exec() 77, 611
_os_exit() 35, 621
_os_findpd() 77, 623
_os_firq() 59, 625
_os_fmod() 69, 627
_os_fork() 77, 628
_os_forkm() 77, 633
_os_get_blkmap() 635
_os_get_ioproc() 49, 643
_os_get_mdp() 70, 644
_os_get_moddir() 70, 645
_os_get_module() 645
_os_get_prtbl() 77, 647
_os_getdl() 49, 637
_os_getime() 87, 638
_os_getpd() 66, 646
_os_getstat() 40, 77, 648
_os_getsys() 40, 77, 649
_os_gprdsc() 77, 651
_os_gregorian() 88, 652
_os_gs_cpypd() 40, 654
_os_gs_cstats() 40, 655
_os_gs_devnm() 40, 656
_os_gs_devtyp() 40, 657
_os_gs_dopt() 40, 659
_os_gs_dsize() 40, 660
_os_gs_edt() 40, 661
_os_gs_eof() 40, 662
_os_gs_fd() 40, 663
_os_gs_fdaddr() 40, 664
_os_gs_fdinf() 40, 665
_os_gs_luopt() 41, 668
_os_gs_parity() 41, 669
_os_gs_popt() 41, 670
_os_gs_pos() 41, 671
_os_gs_ready() 41, 673
_os_gs_size() 41, 674
_os_id() 77, 675

_os_initdata() 70, 678
_os_intercept() 56, 679
_os_ioconfig 681
configure 681
_os_iodel() 49, 682
_os_ioexit() 49, 683
_os_iofork() 49, 684
_os_irq() 59, 686
_os_julian() 88, 690
_os_link() 70, 697
_os_linkm() 70, 699
_os_load() 70, 700
_os_loadp() 70, 702
_os_makdir() 49, 704
_os_makmdir() 70, 706
_os_mem() 63, 707
_os_mkmodule() 708
_os_modaddr() 70, 710
_os_move() 711
_os_nproc() 78, 712
_os_open() 49, 713
_os_permit() 64, 716
_os_perr() 66, 718
_os_protect() 64, 720
_os_prsnam() 66, 722
_os_rdalst() 49, 725
_os_read() 49, 726
_os_readln() 49, 728
_os_rellk() 80, 729
_os_rte() 59, 731
_os_rtnprc() 78, 732
_os_salarm_atime() 56, 735
_os_salarm_cycle() 56, 738
_os_salarm_delete() 57, 742, 743
_os_salarm_reset() 57, 744
_os_salarm_set() 57, 745
_os_scache() 66, 747
_os_seek() 39, 753
_os_sema_init() 57, 754
_os_sema_p() 57, 755
_os_sema_term() 57

_os_sema_v() 57, 757
_os_send() 57, 758
_os_setcrc() 70, 759
_os_setime() 88, 760
_os_setpr() 78, 764
_os_setstat() 41, 765
_os_setsvc() 66, 766
_os_setsys() 41, 770
_os_setuid() 66, 771
_os_sgetstat() 41, 772
_os_sgs_devnm() 41, 773
_os_sigmask() 58, 777
_os_sigreset() 58, 779
_os_sigrs() 58, 781
_os_sleep() 78, 782
_os_slink() 70, 786
_os_slinkm() 71, 788
_os_srqmem() 64, 790
_os_srtmem() 64, 794
_os_ss_attr() 41, 795
_os_ss_break() 41, 796
_os_ss_cache() 42, 797
_os_ss_close() 42, 798
_os_ss_dcoff() 42, 799
_os_ss_dcon() 42, 800
_os_ss_dopt() 42, 801
_os_ss_dsrts() 42, 802
_os_ss_enrts() 42, 803
_os_ss_erase() 42, 804
_os_ss_fd() 42, 805
_os_ss_fillbuff() 42
_os_ss_flushmap() 42, 807
_os_ss_hdlink() 42, 808
_os_ss_lock() 42, 810
_os_ss_luopt() 43, 811
_os_ss_popt() 43, 813
_os_ss_relea() 43, 814
_os_ss_rename() 43, 815
_os_ss_reset() 43, 816
_os_ss_reten() 817
_os_ss_rfm() 43

_os_ss_sendsig() 43, 819
_os_ss_size() 43, 820
_os_ss_skip() 43, 821
_os_ss_skipend() 43, 822
_os_ss_ticks() 43, 823
_os_ss_wfm() 43, 824
_os_ss_wtrack() 43, 825
_os_strap() 66, 827
_os_suspend() 78, 831
_os_sysdbg() 73, 832
_os_sysid() 67, 833
_os_tlink() 71, 840
_os_tlinkm() 71, 842
_os_tranpn() 843
_os_uacct() 67, 848
_os_unlink() 71, 849
_os_unload() 71, 850
_os_vmodul() 71, 851
_os_wait() 78, 854
_os_waitlk() 80, 857
_os_write() 50, 859
_os_writeln() 50, 861
_os9_alarm_atdate() 53, 504
_os9_alarm_atjul() 53, 507
_os9_allbit() 65, 514
_os9_allpd() 76, 515
_os9_aproc() 76, 520
_os9_create() 48, 547
_os9_delbit() 66, 554
_os9_dfork() 77, 567
_os9_ev_wait() 56, 606
_os9_ev_waitr() 56, 609
_os9_findpd() 66, 624
_os9_getime() 88, 640
_os9_gs_cdfd() 40, 653
_os9_gs_free() 41, 667
_os9_gspump() 77, 672
_os9_id() 78, 677
_os9_ioqueue() 49, 685
_os9_irq() 59, 688
_os9_panic() 66, 715

_os9_retpd() 66, 730
_os9_salarm_atdate() 56, 733
_os9_salarm_atjul() 56, 736
_os9_salarm_cycle() 57, 740
_os9_salarm_set() 57, 746
_os9_schbit() 66, 751
_os9_setime() 88, 762
_os9_setsvc() 66, 768
_os9_sleep() 78, 784
_os9_srqmem() 64, 792
_os9_ss_open() 43, 812
_os9_strap() 67, 829
_os9_translate() 64, 846
_os9_vmodul() 71, 852
_parsepath() 67, 106, 867
_prgname() 67, 107, 879
_prsnam() 67, 107, 882
_REAL 140
set<name>() 916
_setcrc() 71, 108, 927
_setsys() 73, 108, 942
_srqcsiz 966
_srqmem() 64, 109, 968
_srqsiz 968
_srtmem() 64, 109, 970
_ss_attr() 44, 109, 971
_ss_dcoff() 44, 109, 975
_ss_dcon() 44, 109, 976
_ss_dsrts() 44, 110, 977
_ss_enrts() 44, 107, 978
_ss_lock() 44, 107, 979
_ss_opt() 44, 107, 981
_ss_pfd() 44, 107, 983
_ss_rel() 44, 108, 985
_ss_rest() 44, 108, 986
_ss_size() 44, 108, 987
_ss_ssig() 44, 108, 988
_ss_tiks() 44, 108, 990
_ss_wtrk() 44, 108, 992
_stacksiz() 67, 994
_strass() 67, 108, 997

_subcall() 71, 1029
_sysdate() 88, 109, 1033–??, 1034–1035
_sysdbg() 73, 109, 1036
_toascii() 38, 1055
_tolower() 38, 1056
_toupper() 38, 1058

Numerics

80x86-specific functions
  get
    current task segment 89, 306
    global descriptor pointer 89, 311
  get_current_tss() 89, 306
  get_excpt_base() 89, 310
  get_gdtr() 89, 311
  list of 89
  make
    global descriptor table entry 89, 439
    interrupt descriptor table entry 89, 440
  make_gdesc() 89, 439
  make_idesc() 89, 440
  return exception table base address 89, 310
  set
    exception table base address 89, 929
    global descriptor pointer 89, 930
  set_excpt_base() 89, 929
  set_gdtr() 89, 930

A

abort() 35, 101, 114
  abnormal program termination 114
abs() 60, 115
absolute value
  abs 115
  compute 415
  fabs() 236
  floating 236
  labs() 415

return 415
access permissions
change for module directory 537
access() 102, 116
accounting 848
acos() 60, 117
acquire
ownership of resource lock 502
conditional 525
activate next process waiting to acquire lock 857
active process queue
_os_aproc 519
_os9_aproc() 520
insert process 519, 520
add
device to IRQ table 686, 688
add filename to root directory variable
buildpath() 149
address translation function
system_addr() 1038
user_addr() 1069
alarm() 118
alarm.h 14
alarms
_os_alarm_atime() 506
_os_alarm_cycle() 509
_os_alarm_delete() 510
_os_alarm_reset() 511
_os_alarm_set() 512
_os_salarm_atime() 735
_os_salarm_cycle() 738
_os_salarm_delete() 742, 743
_os_salarm_reset() 744
_os_salarm_set() 745
_os9_alarm_atdate() 504
_os9_alarm_atjul() 507
_os9_salarm_atdate() 733
_os9_salarm_atjul() 736
_os9_salarm_cycle() 740
_os9_salarm_set() 746
alm_atdate() 120

alm_atjul() 122
alm_cycle() 124
alm_delete() 126
alm_set() 127
delete 126
execute system-state subroutine 740
remove
cyclic alarm 510
pending request 126, 510, 742, 743
reset
alarm clock 744
existing request 511
send
after specified interval 746
at Gregorian date/time 733
at Julian date/time 736
cyclic 740
signal after specified interval 127, 512
signal after specified seconds 118
signal at Gregorian date/time 120, 504
signal at Julian date/time 122, 507
signal at specific time 124, 506, 509, 735
system-state subroutine 733, 736, 746
set alarm clock 738, 745
alias
_os_rdalst() 725
copy system alias list 725
alloca() 119
allocate
_lcalloc() 416
colored memory 966
fixed-length memory block 515
low-overhead 416
memory from arena 423, 444
process descriptor 517
resource lock descriptor 549
storage for array 151, 416
task 516
allocate bytes of space in stack frame 119
alm_atdate() 52, 102, 120
alm_atjul() 52, 103, 122

alm_cycle() 52, 103, 124
alm_delete() 52, 103, 126
alm_set() 52, 103, 127
alpha
  to floating conversion 137
  to integer conversion 138
  to long conversion 139
  to numeric translation 140
alphanumeric 376
append mode
  file position indicator 99
arc
  cosine 117
  sine 131
  tangent 134, 135
array
  allocate storage 151
  search 147
asctime() 87, 129
asin() 60, 131
assert() 65, 91, 132
assert.h 14
AST 84
atan() 60, 134
atan2() 60, 135
atexit() 35, 136
atof() 37, 137
atoi 138
atoi() 37
atol() 37, 139
aton() 140
aton.h 15
attach
  device 143, 521
  path to file pointer 239
attach() 45, 103, 143

B

base-ten logarithm 431
bcmp() 145

bcopy() 146
binary file
    create temporary 1053
    tmpfile() 1053
bit index find
    ffs() 244
bit map
    _os9_allbit() 514
    _os9_delbit() 554
    _os9_schbit() 751
    deallocate 554
    flush cached information 807
    search for free area 751
    set bits 514
break
    serial connection 796
    string into tokens 1019
broken-down time
    convert to
        calendar time 462
        string format 129
    structure 1031
bsearch() 80, 147
buildpath() 149
bypass macro replacement 8
bzero 150

C

cache
    control 523, 747
    status
        _os_gs_cstats() 655
        get information 655
cache.h 15
calculate
    module CRC 181
    parity of file descriptor 669
calendar time 341
call
    register function 136

subroutine module 1029
system debugger 832, 1036
calloc() 62, 101, 151
capability ID
check 1045
get 1046, 1047
tgetflag() 1045
tgetnum() 1046
tgetstr() 1047
capability string
output 1060
tputs() 1060
cbrt() 153
cdi.h 15
CDT 84
ceil() 60, 154
ceiling function 154
CET 84
chain() 155
chainc() 76, 103, 155
change
current constant data pointer 158
current data directory 160
current execution directory 166
current static storage pointer 159
file access permissions 162
module directory 534
owner of file 164
permissions of module directory 537
system global variable 942
working directory 530
change_static() 65, 159
CHAR_MAX 425, 426
character
get from file 245
get from I/O address 360
get next from file/stdin 303
multibyte 446, 448, 450, 1083, 1085
output to file 267
push back 1064
put next to file/stdout 883

rindex() 907
search for 361, 907
standard white space 405
unget 1064
ungetc() 1064
write to I/O address 864
character classification
_isascii() 36, 380
_isjis_kana() 36, 387, 389, 393
_issjis1() 37, 401
_issjis2() 37, 403
isalnum() 36, 376
isalph() 378
isalpha() 36
isascii() 380
iscntrl() 36, 383
isdigit() 36, 385
isgraph() 36, 391
islower() 36, 395
isprint() 36, 397
ispunct() 36, 399
isspace() 37, 405
isupper() 37, 407
isxdigit() 37, 409
list of functions 36
test for
alphabetic 36, 378
alphanumeric 36, 376
ASCII 36, 380
control code 36, 383
digit 36, 385
Hankaku-kana 36, 387, 389, 393
hexadecimal 37, 409
Kanji 37, 401, 403
lowercase 36, 395
printable character 36, 397
printing character 36, 391
punctuation character 36, 399
uppercase 37, 407
white space 37, 405
character conversion

_aton() 37, 140
_atou() 38, 142
_dtoa() 38, 194
_toascii() 38, 1055
_tolower() 38, 1056
_toupper() 38, 1058
alpha to
floating 37, 137
integer 37, 138
long 37, 139
numeric 37, 140
unsigned 38, 142
atof() 37, 137
atoi() 37, 138
atol() 37, 139
convert
double to ASCII 38
to lowercase 38, 1056, 1057
to uppercase 38, 1058, 1059
double to ASCII 194
list of functions 37
toascii() 1055
tolower() 38, 1057
toupper() 38, 1059
translate 38, 1055
chdir() 103, 160
check
buffered file for
end of file 241
error condition 242
for
deadlock situation 552
end of file 346
use of I/O module 682
memory block's accessibility 532
terminal capability presence 1045
chmod() 103, 162
chown() 103, 164
chxdir() 46, 104, 166
clear
end of file condition 168

error condition 170
process signal queue 536
clearerr() 168
clearerr() 46, 170
clib.l 10, 102
clock() 87, 171
era for 102
CLOCKS_PER_SEC 124, 127, 171
close
file 237
named directory stream 174
path 172, 535, 798
close() 46, 104, 172
closedir() 104, 174
colored memory
allocate 966
load module into 464
modcload() 464
srqcmem() 966
compare
memory 453
names 538
strings 175, 1000, 1001, 1011
compute
absolute value 415
floating point remainder 255
ldiv() 419
quotient 193, 419
remainder 193, 419
string length 1015
constant data
change current pointer 158
control
locale 935
process' signal handling 947
convert
_dtoa() 194
alpha to
floating 137
integer 138
long 139

numeric 140
unsigned 142
broken-down time 129
broken-down to calendar time 462
calendar to
Greenwich Mean Time 341
local time 429
string format 189
character to
lowercase 1056, 1057
uppercase 1058, 1059
date/time to
Gregorian value 342
Julian value 411
double to ASCII 194
input string 279
local to Greenwich Mean Time 1031
localtime() 429
mbstowcs() 448
mktime() 462
sequence of multibyte characters 448
string to
double 1017
long 1021, 1024
wide to multibyte characters 1083, 1085
convert characters
carriage return to linefeed 188
linefeed to carriage return 422
copy
_os_rdalst() 725
contents of path descriptor 654
external memory 179, 542
memory 454
old OS-9 strings 1008
string 1002, 1012
system alias list 725
cos() 60, 177
cosh() 60, 178
cosine 177
arc 117
hyperbolic 178

cpu.l 11
CRC
  _os_crc() 543
  _os_setcrc() 759
  calculate 181
  check value 543
  crc() 181
  generate 543, 759
crc() 68, 104, 181
CRCCON 543
creat() 46, 104, 184
create
  data module 457, 550, 708
  device alias 513
  directory 438, 459, 704
  event 579
  file 184, 186, 547
  hard link 808
  module 441
  module directory 706
  path to new file 545
  process 628, 630, 633
  resource lock descriptor 549
  temporary binary file 1053
  unique file name 461
create temporary file names 1042
create() 46, 102, 186
crtolf() 188
CST 84
cstart.r 10
ctime() 87, 189
ctype.h 15
cube root
  cbrt() 153
cursor movement
  get capability 1049
  tgoto() 1049

D

data

_gs_rdy() 353
_os_move() 711
carrier detect 799, 800
move 711
test for available 353
data module
_os_datmod() 550
_os_mkmodule() 708
create 550, 708
specify color type 708
deallocate
bits in bit map 554
memory 271
process descriptor 559, 732
debugged program 561
_os_dexec() 561
_os_dexit() 564
_os_dfork() 565
_os9_dfork() 567
execute 561
exit 564
fork 565, 567
terminate 564
debugger control
_os_dforkm() 569
fork process 569
delete
existing event 581
existing lock descriptor 557
existing module directory 558
file 555, 903, 1066, 1067
detach device 191, 560
detach() 46, 102, 191
determine
end of file 346
file accessibility 116
number of bytes in multibyte character 450
parent process ID 319
path number from file 249
process ID 318
size of unused stack area 272

string length 1009
user ID 338
device 521, 648
_gs_devn() 344
_os_alias() 513
_os_close() 535
_os_detach() 560
_os_getstat() 648
_os_gs_devnm() 656
_os_gs_devtyp() 657
_os_gs_dopt() 659
_os_gs_dsize() 660
_os_irq() 686
_os_open() 713
_os_read() 726
_os_setstat() 765
_os_sgs_devnm() 773
_os_ss_dopt() 801
_os_ss_relea() 814
_os_write() 859
_os9_gs_free() 667
_os9_irq() 688
_ss_enrts() 978
_ss_rel() 985
_ss_rest() 986
add to IRQ table 686, 688
attach to system 143, 521
close path 535
create alias 513
detach 560
detach() 191
disable RTS line 977
enable RTS line 978
get name 344
get size of SCSI device 660
open 713
read data from 726
read path options 659
release 814, 985
remove 191, 560
from IRQ table 686, 688

restore 986
return
amount of free space 667
name 656, 773
type 657
set
path options 801
status 765
write data 859
device controller
ioctl() 367
difftime() 87, 192
dir.h 16
direct.h 16
directory
_os_altmdir() 518
_os_chdir() 530
_os_chmdir() 534
_os_cmdperm() 537
_os_delmdir() 558
_os_fmod() 627
_os_get_mdp() 644
_os_get_moddir() 645
_os_makdir() 704
_os_makmdir() 706
change
current data 160
current execution 166
current module directory 534
permissions on module directory 537
working directory 530
chdir() 160
chxdir() 166
close named stream 174
closedir() 174
create 438, 459, 704
create module directory 706
delete module directory 558
establish alternate module directory 518
find module directory entry 627
get

alternate module directory pathlist 644
copy of module directory 645
current module directory pathlist 644
get current working 340
initialize 704
makdir() 438
make 704
mknod() 459
open 501
opendir() 501
readdir() 893
reset position of directory stream 906
return
current location 1041
pointer to 893
rewinddir() 906
seekdir() 913
set
alternate working module directory 518
position of next readdir 913
telldir() 1041
disable
RBF caching 797
RTS line 802, 977
div() 60, 193
double to ASCII conversion 194
driver
_os_ss_close 798
_os9_ss_open() 812
notify of open path 812
notify that path has closed 798
dtoa() 194
dup() 103, 196
dup2() 197
duplicate path 196, 197, 571

E

ebrk() 62, 103, 198
EDT 84
EET 84

enable
RBF caching 797
RTS line 803, 978
end references to current variable parameter list 1071
endpwent() 200
enter
I/O queue 685
process in active process queue 520
environment
change or add value 885
environment functions
_exit() 35, 232
_os_exit() 35, 621
abnormal program termination 35, 114
abort() 35, 114
atexit() 35, 136
exit() 35, 233
get value for environment name 35, 307
getenv() 35, 307
listed 35
register function to call at normal program termination 35, 136
shell command execution 35, 1037
system() 35, 1037
task termination 35, 232, 233
terminate calling process 35, 621
environment name
get value for 307
environment names
common set of 101
environment variables
TERM 1043
TERMCAP 1043
TZ 84, 102
ERANGE 92
erase tape 804
errno 92
value to which macro is set on failure 100
errno.h 16
error
_errmsg() 201
_os_perr() 718

_os_strap() 827
_os9_strap() 829
check buffered file for error condition 242
clear 170
display message 201
ferror() 242
format of messages 1004
map number 871
perror() 871
prerr() 877
print message 201, 718, 877
set trap handler 827, 829
error message strings
contents of 101
error messages
format of 100
EST 84
Euclidean distance function 357
events
_ev_creat() 203
_ev_delete() 205
_ev_info() 206
_ev_link() 208
_ev_pulse() 209
_ev_read() 211
_ev_set() 213
_ev_setr() 215
_ev_signal() 217
_ev_unlink() 219
_ev_wait() 221
_ev_waitr() 223
_os_ev_allclr() 573
_os_ev_allset() 575
_os_ev_anyclr() 576
_os_ev_anyset() 577
_os_ev_change() 578
_os_ev_creat() 579
_os_ev_delete() 581
_os_ev_info() 582
_os_ev_link() 585
_os_ev_pulse() 586

_os_ev_read() 588
_os_ev_set() 589
_os_ev_setand() 591
_os_ev_setor() 593
_os_ev_setr() 595
_os_ev_setxor() 597
_os_ev_signal() 599
_os_ev_tstset() 601
_os_ev_unlink() 603
_os_ev_wait() 604
_os_ev_waitr() 607
_os9_ev_wait() 606
_os9_ev_waitr() 609
create 203, 579
decrement link count 219
delete 205, 581
get information 206
link to existing event 208, 585
read event value 588
read without waiting 211
return event information 582
set event variable 213, 589, 591, 593, 597
relative 215, 595
signal event occurrence 209, 213, 215, 217, 586, 589, 591, 593, 595, 597, 599
unlink 219, 603
wait for
bits to clear 573
event to occur 221, 573, 575, 576, 577, 578, 601, 604, 606
relative event 223, 607, 609
events.h 17
examine system global variable 336, 649, 942
example
error file 718
for generating CRC 182
initialization table 766, 768, 827
of _errmsg() 202
of _get_process_desc() 320
of _julian() 412
of _os_exec() 615

of _os_getsys() 650
of _sysdate() 1035
of fopen() 259
of index() 361
of intercept() 366
of os9exec() 620
of os9kexec() 696
of rindex() 907
of strtok() 1020
of vfprintf() 1074
exception table
set base address 929
set_excpt_base() 929
exec interfaces
execl() 225
execle() 227
execv() 229, 230
execvp() 231
execl() 225
execle() 227
execute 561
module 526, 528
new module 155
shell command 1037
system-state subroutine 746
at Gregorian date/time 733
at Julian date/time 736
execv() 229, 230
execvp() 231
exit debugged program 564
exit() 35, 233
status returned by 101
exp() 60, 235
expand
value to integral constand expression 497
exponential function 235
extend data memory segment 909
external memory 542

F

fabs() 60, 236
fclose() 46, 237
fdopen 104
fdopen() 46, 104, 239
feof() 47, 241
ferror() 47, 242
fflush() 47, 243
ffs() 244
fgetc() 47, 245
fgetpos() 38, 100, 246
fgetpwent() 247
fgets() 47, 248
file 883
    _fileno() 249
    _gs_pos() 352
    _gs_size() 355
    _os_close() 535
    _os_create() 545
    _os_delete() 555
    _os_getstat() 648
    _os_gs_fdaddr() 664
    _os_gs_pos() 671
    _os_gs_size() 674
    _os_open() 713
    _os_read() 726
    _os_setstat() 765
    _os_ss_attr() 795
    _os_ss_hdlink() 808
    _os_ss_rename() 815
    _os_ss_size() 820
    _os_write() 859
    _os9_create() 547
    _ss_attr() 971
    _ss_size() 987
    affect of opening multiple times 100
    attach path to file pointer 239
    change
        access permissions 162
        attributes 971

name 815, 904
owner 164
size 987
chmod() 162
chown() 164
close 237
close path 535
creat() 184
create 184, 186, 545, 547
temporary binary file 1053
unique name 461
create() 186
delete 555, 903, 1066, 1067
determine
accessibility 116
path number 249
size 355
fclose() 237
fdopen() 239
fflush() 243
fgetc() 245
fgetpos() 246
fgets() 248
fileno() 249
fix buffer 926
flush buffer 243
fopen() 256
fputc() 267
fputs() 268
fread() 269
freopen() 276
fseek() 285
fsetpos() 287
ftell() 289
fwrite() 290
generate unique name 1054
get
character from file 245
current position 246, 352, 671
current size 355
file descriptor block address 664

status 332, 648
string from 248, 331
gets() 331
getstat() 332
getw() 339
load module from 700
lseek() 436
make hard link to 808
mktemp() 461
open 256, 498, 713
open() 498
output
character to 267
string to 268, 886
word to 887
position pointer 436
print to 1073, 1075, 1077
read
characters from 331
data from 269, 726
word 339
remove 903
remove() 903
rename 815, 904
rename() 904
re-open 276
report pointer position 289
reposition pointer 285
return
current size 674
path number 184, 186
pointer to zero 905
rewind() 905
set
attributes 795, 971
current position 287
current size 987
size 820
status 765, 940
setbuf() 926
setstat() 940

tmpfile() 1053
tmpnam() 1054
unlink 1066, 1067
unlink() 1066
unlinkx() 1067
valid access modes 239, 256
vfprintf() 1073
vprintf() 1075
vsprintf() 1077
write
data 290, 859
string to 268
unwritten data 243
file descriptor
_gs_gfd() 348
_os_gs_fd() 663
_os_gs_fdaddr() 664
_os_gs_fdinf() 665
_os_gs_parity() 669
_os_ss_fd() 805
_os9_gs_cdfd() 653
_ss_pfd() 983
calculate parity 669
get block address 664
get sector 348, 665
put sector 983
read sector 663
return 653
write sector 805
file names
composing valid 99
file pointer
report position 289
reposition 285, 436
logical 753
return to zero 905
file positioning functions
_os_seek() 39, 753
fgetpos() 38, 246
fseek() 38, 285
fsetpos() 38, 287

ftell() 38, 289
get current position in file 38, 246
list of 38
report file pointer position 38, 289
reposition
file pointer 38, 285
logical file pointer 39, 753
return file pointer to zero 39, 905
rewind() 39, 905
set current file position 38, 287
fileno() 249
fill memory 456
find
fixed block of memory 624
module 710
module directory entry 627
path descriptor 646
process descriptor 623
temporal difference 192
terminal name
isatty() 382
find first set bit
ffs() 244
findnstr() 81, 104, 250
findstr() 81, 104, 252
fix file buffer 926
float.h 17
floating absolute value 236
floor function 254
floor() 60, 254
flush
cached bit map information 807
file buffer 243
fmod() 60, 92, 255
fopen() 47, 256
fork process under control of debugger 565, 567, 569
format of
error messages 100
Gregorian value 342
time and date values 342
formatted output 260, 880, 962

fprintf() 47, 100, 260
fputc() 47, 267
fputs() 47, 268
fread() 47, 269
free() 62, 271
freemem() 272
freopen() 47, 276
frexp() 61, 278
from/to process
  create communication pipe 874
fscanf() 47, 100, 279
fseek() 38, 285
fsetpos() 38, 287
fstat() 288
ftell() 38, 100, 289
funcs.h 17
function
  defined 8
fwrite() 47, 290

G

generate
  CRC 181, 543
  unique valid file name 1054
  valid CRC 759
get
  alternate module directory pathlist 644
  cache status information 655
  calendar time 1051
  character from
    file 245
    I/O address 360
  copy of
    module directory 645
    process descriptor block table 647
  current
    constant data pointer 305
    file position 352, 671
    file size 355
    module directory pathlist 644

position in file 246
static storage pointer 335
task segment 306
cursor movement capability 1049
device
name 344
status 648
event information 206
exception table base address 310
external memory 198
file descriptor
block address 664
sector 348, 665
file status 332, 648
free memory block map 635
global descriptor pointer 311
Gregorian date 652
I/O interface edition number 661
integer from I/O address 363
Julian date 690
module directory entry 313
module name 879
next character from file/stdin 303
ownership of resource lock 502
parameter in variable parameter list 1070
path options 350
pointer to I/O process descriptor 541, 643
process descriptor copy 320, 651
process ID 675, 677
process table entry 322
processor time 171
size of
SCSI devices 660
stack used 994
status 648
string
from file 248
length 1003
system
date 638, 640
global variables 336

I/O device list head pointer 637
time 312, 638, 640
Termcap entry 1043
terminal capability 1047
terminal capability ID 1046
user ID 338, 675, 677
value for environment name 307
value of registers 292
word from I/O address 369
get file status
fstat() 288
stat() 995
get_const() 305
get_current_tss() 89, 306
get_excpt_base() 89, 310
get_gdtr() 89, 311
get_static() 65, 335
getc() 47, 303
getchar() 303
getenv() 35, 101, 307
getgid() 309
getime() 87, 105, 312
getnextpwnam 314
getopt() 315
getpid() 76, 105, 318
getppid() 319
getpw() 323
getpwent() 324
getpwnam() 327
getpwuid() 329
gets() 48, 331
GetStat
_gs_devn() 39, 344
_gs_eof() 39, 346
_gs_gfd() 39, 348
_gs_opt() 39, 350
_gs_pos() 39, 352
_gs_rdy() 39, 353
_gs_size() 39, 355
_os_getstat() 40, 648
_os_getsys() 40, 649

_os_gs_cpypd() 40, 654
_os_gs_cstats() 40, 655
_os_gs_devnm() 40, 656
_os_gs_devtyp() 40, 657
_os_gs_dopt() 40, 659
_os_gs_dsize() 40, 660
_os_gs_edt() 40, 661
_os_gs_eof() 40, 662
_os_gs_fd() 40, 663
_os_gs_fdaddr() 40, 664
_os_gs_fdinf() 40, 665
_os_gs_luopt() 41, 668
_os_gs_parity() 41, 669
_os_gs_popt() 41, 670
_os_gs_pos() 41, 671
_os_gs_ready() 41, 673
_os_gs_size() 41, 674
_os_sgetstat() 41, 772
_os_sgs_devnm() 41
_os9_gs_cdfd() 40, 653
_os9_gs_free() 41, 667
calculate parity of file descriptor 41, 669
check for end of file 39, 346
copy contents of path descriptor 40, 654
examine system global variable 40, 649
get
    cache status information 40, 655
    current file position 39, 41, 352, 671
    current file size 39, 355
    device name 39, 344
    device status 40, 648
    file
        descriptor block address for open file 40, 664
        descriptor sector 39, 40, 348
        status 40, 332, 648
    I/O interface edition number 40, 661
    path options 39, 350
    size of SCSI device 40, 660
    specified file descriptor sector 665
GetStat call using system path number 41, 772
getstat() 332

list of functions 39
read
device path options 40, 659
file descriptor sector 40, 663
logical unit options 41, 668
path descriptor options section 41, 670
return
amount of free space on device 41, 667
current file size 41, 674
device name 40, 41, 656
device type 40, 657
file descriptor 40, 653
test
for data available 39, 353
for data ready 41, 673
for end of file 40, 662
getstat() 105, 332
gettimeofday() 337
getuid() 65, 105, 338
getw() 48, 105, 339
getwd() 340
global descriptor
make entry in table 439
make_gdesc() 439
set pointer 930
set_gdtr() 930
globals 12
GMT 84
gmtime() 87, 341
Greenwich Mean Time 341, 1031
Gregorian
_gregorian() 342
_os_gregorian() 652
convert to 342
group identification
getgid() 309

H

hard link
_os_ss_hdlink() 808

create 808
header files 14
alarm.h 14
assert.h 14
aton.h 15
cache.h 15
cdi.h 15
ctype.h 15
dir.h 16
direct.h 16
errno.h 16
events.h 17
float.h 17
funcs.h 17
io.h 18
ioctl.h 18
ioedt.h 18
limits.h 18
locale.h 18
lock.h 19
math.h 19
memory.h 20
moddir.h 20
modes.h 21
module.h 22, 543
os9def.h 23
os9time.h 23
process.h 24
procid.h 25
rbf.h 25
regexp.h 25
regs.h 26
sbf.h 26
scf.h 27
semaphore.h 27
setjmp.h 27
setsys.h 27
settrap.h 27
sg_codes.h 28
sgstat.h 28
signal.h 29

stdarg.h 29
stddef.h 29
stdio.h 30
stdlib.h 31
string.h 32
strings.h 32
svctbl.h 32
sysglob.h 33
termcap.h 33
termio.h 33
time.h 33, 34
un.h 34
hyperbolic
cosine 178
sine 959
tangent 1040
hypot() 61, 106, 357

I

I/O address
get
character from 360
integer from 363
word from 369
inc() 360
inl() 363
inw() 369
outc() 864
outl() 865
outw() 866
write
character to 864
integer to 865
word to 866
I/O device
_os_getdl() 637
attach 521
get system pointer for list 637
I/O device functions
get from I/O address

character 45, 360
integer 45, 363
word 45, 369
inc() 45, 360
inl() 45, 363
inw() 45, 369
list of 45
outc() 45, 864
outl() 45, 865
outw() 45, 866
write to I/O address
character 45, 864
integer 45, 865
word 45, 866
I/O functions
_cleareof() 46, 168
_fileno() 47, 249
_os_alias() 48, 513
_os_attach() 48, 521
_os_chdir() 48, 530
_os_close() 48, 535
_os_cpy_ioproc() 48, 541
_os_create() 48, 545
_os_delete() 48, 555
_os_detach() 49, 560
_os_dup() 49, 571
_os_get_ioproc() 49, 643
_os_getdl() 49, 637
_os_iodel() 49, 682
_os_ioexit() 49, 683
_os_iofork() 49, 684
_os_makdir() 49, 704
_os_open() 49, 713
_os_rdalst() 49, 725
_os_read() 49, 726
_os_readln() 49, 728
_os_write() 50, 859
_os_writeln() 50, 861
_os9_create() 48, 547
_os9_ioqueue() 49, 685
attach

new device to system 48
path to file pointer 46, 239
to device 45, 143, 521
attach() 45, 143
change
current execution directory 46, 166
working directory 48, 530
check
buffered file for
end of file 47, 241
error condition 47, 242
for use of I/O module 49, 682
chxdir() 46, 166
clear
end of file condition 46, 168
error condition 46, 170
cleareof() 168
clearerr() 46, 170
close
file 46, 237
path 46, 48, 172, 535
close() 46, 172
copy system alias list 49, 725
creat() 46, 184
create
device alias 48, 513
directory 48, 438, 459
file 46, 184, 186
path to new file 48, 545, 547
temporary binary file 51, 1053
unique file name 48, 461
create() 46, 186
delete file 48, 555
detach device 46, 191
detach() 46, 191
determine path number from file 47, 249
duplicate path 49, 571
enter I/O queue 49, 685
fclose() 46, 237
fdopen() 46, 239
feof() 47, 241

ferror() 47, 242
fflush() 47, 243
fgetc() 47, 245
fgets() 47, 248
fileno() 249
fix file buffer 50, 926
flush file's buffer 47, 243
fopen() 47, 256
formatted output 47, 50, 51, 260, 880, 962
fprintf() 47, 260
fputc() 47, 267
fputs() 47, 268
fread() 47, 269
freopen() 47, 276
fscanf() 47, 279
fwrite() 47, 290
get
  character 245, 303
  character from
    file 47
  pointer to I/O process descriptor 48, 541, 643
  string from file 47, 48, 248, 331
  system I/O device list head pointer 49, 637
get character from
  file 47
get pointer to I/O process descriptor 49
getc() 47, 303
getchar() 303
gets() 48, 331
getw() 48, 339
initialize for
  float output 50
  longs output 50
input string conversion 47, 51, 279, 911, 973
input strings conversion 50
list of 45
makdir() 48, 438
make new directory 49, 704
mknod() 48, 459
mktemp() 48, 461
open

directory 48, 501
file 47, 48, 256, 498
path to file or device 49, 713
open() 48, 498
opendir() 48, 501
output to file
character 47, 267
string 47, 50, 268, 886
word 50, 887
pffinit() 50, 872
pflinit() 50, 873
print to
file 51, 1073, 1075, 1077
standard output 51
string 51
printf() 50, 880
put next character 50, 883
putc() 50, 883
putchar() 883
puts() 50, 886
putw() 50, 887
read
bytes from path 50, 891, 894
data from device 49, 726
data from file 47, 49, 269, 726
text line with editing 49, 728
word from file 48, 339
read() 50, 891
readln() 50, 894
remove
device from system 49, 560
file 50, 903
remove() 50, 903
rename file 50
rename() 50
re-open file 47, 276
scanf() 50, 911
set up
buffer for I/O stream 51, 945
I/O for new process 49, 684
setbuf() 50, 926

setvbuf() 51, 945
sprintf() 51, 962
sscanf() 51, 973
terminate I/O for exiting process 49, 683
tmpfile() 51, 1053
unget character 51, 1064
ungetc() 51, 1064
unlink file 51, 1066, 1067
unlink() 51, 1066
unlinkx() 51, 1067
vfprintf() 51, 1073
vprintf() 51, 1075
vsprintf() 51, 1077
write
bytes to path 51, 1087, 1089
data to device 50, 859
data to file 47, 50, 290, 859
line of text with editing 50, 861
write() 51, 1087
writeln() 51, 1089
I/O interface edition number
_os_gs_edt() 661
get 661
I/O module
_os_iodel() 682
check for use of 682
I/O process descriptor
_os_cpy_ioproc() 541
_os_get_ioproc() 643
get pointer to 541, 643
I/O queue
_os9_ioqueue() 685
enter 685
I/O stream
set buffer for 945
setvbuf() 945
ibrk() 62, 106, 358
inc() 45, 360
index() 81, 107, 361
indicate event occurrence 209, 213, 215, 217, 593
initialize

directory 704
for float output 872
for longs output 873
process descriptor 516, 517
resource lock descriptor 549
service request table 768
static storage 678
variable parameter list 1072
inl() 45, 363
input
failures
defined 282
read from stream 279
string conversion 279, 911, 973
insert process in active process queue 519
install
system state trap handling module 840
user subroutine library module 788
user subroutine module 786
user trap handling module 842
integer
return random 890
write to I/O address 865
integerget from I/O address 363
intercept trap
_os_intercept() 679
set up 679
intercept() 53, 107, 364
interprocess communication
_ev_creat() 52, 203
_ev_delete() 52, 205
_ev_info() 52, 206
_ev_link() 53, 208
_ev_pulse() 53, 209
_ev_read() 53, 211
_ev_set() 53, 213
_ev_setr() 53, 215
_ev_signal() 53, 217
_ev_unlink() 53, 219
_ev_wait() 53, 221
_ev_waitr() 53, 223

_os_alarm_atime() 54, 506
_os_alarm_cycle() 54, 509
_os_alarm_delete() 54, 510
_os_alarm_reset() 54, 511
_os_alarm_set() 54, 512
_os_clrsigs() 54, 536
_os_ev_allclr() 54, 573
_os_ev_allset() 54, 575
_os_ev_anyclr() 54, 576
_os_ev_anyset() 54, 577
_os_ev_change() 54, 578
_os_ev_creat() 54, 579
_os_ev_delete() 55, 581
_os_ev_info() 55, 582
_os_ev_link() 55, 585
_os_ev_pulse() 55, 586
_os_ev_read() 55, 588
_os_ev_set() 55, 589
_os_ev_setand() 55, 591
_os_ev_setor() 55, 593
_os_ev_setr() 55, 595
_os_ev_setxor() 55, 597
_os_ev_signal() 55, 599
_os_ev_tstset() 56, 601
_os_ev_unlink() 56, 603
_os_ev_wait() 56, 604
_os_ev_waitr() 56, 607
_os_intercept() 56, 679
_os_salarm_atime() 56, 735
_os_salarm_cycle() 56, 738
_os_salarm_delete() 57, 742, 743
_os_salarm_reset() 57, 744
_os_salarm_set() 57, 745
_os_sema_init() 57, 754
_os_sema_p() 57, 755
_os_sema_term() 57
_os_sema_v() 57, 757
_os_send 758
_os_send() 57
_os_sigmask() 58, 777
_os_sigrs() 58, 781

_os9_alarm_atdate() 53, 504
_os9_alarm_atjul() 53, 507
_os9_ev_wait() 56, 606
_os9_ev_waitr() 56, 609
_os9_salarm_atdate() 56, 733
_os9_salarm_atjul() 56, 736
_os9_salarm_cycle() 57, 740
_os9_salarm_set() 57, 746
alarm() 118
alm_atdate() 52, 120
alm_atjul() 52, 122
alm_cycle() 52, 124
alm_delete() 52, 126
alm_set() 52, 127
clear process signal queue 54, 536
control process' signal handling 58, 947
create event 52, 54, 203, 579
delete event 52, 55, 205, 581
execute system-state subroutine
    after specified time interval 57, 746
    at Gregorian date/time 56, 733
    at Julian date/time 56, 736
    at specified time interval 56
    every n ticks/second 57, 740
get event information 52, 206
initialize semaphore data structure 57
intercept() 53, 364
link to existing event 53, 55, 208, 585
list of functions 52
mask signals during critical code 58, 777
pause() 58, 869
raise() 58, 889
read event without waiting 53, 55, 211, 588
release semaphore 57
remove pending alarm request 52, 54, 57, 126, 510, 742, 743
reserve semaphore 57
reset alarm
    clock 57, 744
    request 54, 511
resize current process' signal queue block 58, 781

return event information 55, 582
send signal
after specified time interval 52, 54, 127, 512
at Gregorian date/time 52, 53, 120, 504
at Julian date/time 52, 53, 122, 507
at seconds 118
at specified time 54, 506, 735
at specified time intervals 52, 54, 124, 509
to another process 57, 758
to program 58, 889
set
alarm clock 57, 745
cyclic alarm clock 56, 738
event variable 53, 55, 213, 589, 591, 593, 597
process signal handler 53, 364, 679
relative event variable 53, 55, 215, 595
signal intercept trap 56
sigmask() 58, 947
signal event occurrence 53, 55, 209, 213, 215, 217, 586, 589, 591, 593, 595, 597, 599
signal() 58, 950
specify signal handling 58, 950
terminate use of semaphore data structure 57
unlink event 53, 56, 219, 603
unmask signals 58, 777
wait for
all bits defined by mask to become clear 54, 573
event to occur 53, 54, 56, 221, 575, 576, 577, 578, 601, 604, 606
relative event to occur 53, 56, 223, 607, 609
signal 58, 869
interrupt descriptor table
make entry 440
make_idesc() 440
interrupt exception
_os_rte() 731
return from 731
interrupt level
irq_change() 370
irq_disable() 371
irq_enable() 372

irq_maskget() 373
irq_restore() 374
irq_save() 375
mask 371, 373
restore 374
return current 375
return to previous 373
set 370
unmask 372
interrupt manipulation
_os_firq() 59
_os_irq() 59, 686
_os_rte() 59, 731
_os_sigreset() 58
_os9_irq() 59, 688
add device to fast IRQ table 59
add device to IRQ table 59, 686, 688
irq_change() 59, 370
irq_disable() 59, 371
irq_enable() 59, 372
irq_maskget() 59, 373
irq_restore() 59, 374
irq_save() 59, 375
list of functions 59
mask interrupts 59, 371, 373
remove device from fast IRQ table 59
remove device from IRQ table 59, 686, 688
reset signal intercept context stack 58
restore interrupt level 59, 374
return
current interrupt level 59, 375
from interrupt exception 59, 731
set IRQ level 59, 370
unmask interrupts 59, 372
inw() 45, 369
io.h 18
ioctl() 367
ioctl.h 18
ioedt.h 18
IRQ
table

_os_firq() 625
add device
fast 625
remove device
fast 625
IRQ level
irq_change() 370
mask interrupts 371, 373
restore interrupt level 374
return current interrupt level 375
set 370
unmask interrupts 372
IRQ table
_os_irq() 686, 688
add device 686, 688
remove table 686, 688
irq_change() 59, 370
irq_disable() 59, 371
irq_enable() 59, 372
irq_maskget() 59, 373
irq_restore() 59, 374
irq_save() 59, 375
isalnum() 36, 91, 376
isalpha() 36, 378
isascii() 380
isatty() 382
iscntrl() 36, 91, 383
isdigit() 36, 385
isgraph() 36, 391
islower() 36, 91, 395
isprint() 36, 91, 397
ispunct() 36, 399
isspace() 37, 405
isupper() 37, 407
isxdigit() 37, 409

J

Japanese Industrial Standard 401, 403
JST 84
Julian date and time 411

Julian day defined 122

K

kill() 76, 107, 413

L

labs() 61, 415
LC_ALL 428, 935
LC_COLLATE 935, 1001
LC_CTYPE 935
LC_MONETARY 428, 935
LC_NUMERIC 428, 935
LC_TIME 935, 936, 1006
ldexp() 61, 418
ldiv() 61, 419
lftocr() 422
limits.h 18
link 697
    to existing event 208, 585
    to memory module 467, 469, 471, 697, 699, 702
load
    memory module 155, 469, 471, 700, 702
    module into colored memory 464
locale control 935
locale.h 18
localeconv() 65, 425
localtime() 87, 429
locate
    first occurrence of string 1013, 1016
    last occurrence of string 1014
    string 999
lock
    _os_acqlk() 502
    _os_caqlk() 525
    _os_crlk() 549
    _os_ddlk() 552
    _os_dellk() 557
    _os_rellk() 729

_os_waitlk() 857
acquire 502
activate next 857
conditionally acquire ownership of 525
create 549
deadlock situation 552
delete descriptor 557
get ownership 502
out record 810, 979
release ownership 729
lock.h 19
log() 61, 430
log10() 61, 431
logarithm
base-ten 431
natural 430
logical file pointer
_os_seek() 753
reposition 753
logical unit
_os_gs_luopt() 668
_os_ss_luopt() 811
read options 668
write options 811
longjmp() 65, 432
lseek() 107, 436

M

m_tmzone 87, 102
macro
bypass replacement 8
defined 8
makdir() 48, 105, 438
make
directory 704
global descriptor table entry 439
interrupt descriptor table entry 440
make_gdesc() 89, 439
make_idesc() 89, 440
make_module() 68, 105, 441

malloc() 63, 101, 444
map
  error number 871
mask
  interrupts 371, 373
  signals 777
matching failures
  defined 282
math.h 19
mathematical functions
  abs() 60, 115
  acos() 60, 117
  arc
    cosine 60, 117
    sine 60, 131
    tangent 60, 134, 135
  asin() 60, 131
  atan() 60, 134
  atan2() 60, 135
  base-ten logarithm 61, 431
  ceil() 60, 154
  ceiling function 60, 154
  compute
    absolute value 61, 415
    floating point remainder 60, 255
    quotient and remainder 60, 61, 193, 419
  cos() 60, 177
  cosh() 60, 178
  cosine 60, 177
    arc 60, 117
    hyperbolic 60, 178
  div() 60, 193
  Euclidean distance function 61, 357
  exp() 60, 235
  exponential function 60, 235
  fabs() 60, 236
  floating absolute value 60, 236
  floor function 60, 254
  floor() 60, 254
  fmod() 60, 255
  frexp() 61, 278

hyperbolic
cosine 60, 178
sine 61, 959
tangent 61, 1040
hypot() 61, 357
integer absolute value 60, 115
labs() 61, 415
ldexp() 61, 418
ldiv() 61, 419
list of 60
log() 61, 430
log10() 61, 431
modf() 61, 466
multiply float by exponent of 2 61, 418
natural logarithm 61, 430
pow() 61, 876
power function 61, 876
rand() 61, 890
return
parts of real number 61, 466
portions of floating point number 61, 278
random integer 61, 890
set seed for random number generator 61, 965
sin() 61, 958
sine 61, 958
arc 60, 131
hyperbolic 61, 959
sinh() 61, 959
sqrt() 61, 964
square root 61, 964
srand() 61, 965
tan() 61, 1039
tangent 61, 1039
arc 134, 135
hyperbolic 61, 1040
tanh() 61, 1040
mblen() 72, 446
mbstowcs() 448
mbstowsc() 72
mbtowc() 72, 450
MDT 84

MEM_ANY 442, 464, 966
MEM_SYS 442, 464, 966
memchr() 63, 452
memcmp() 63, 453
memcpy() 63, 454
memmove() 63, 455
memory
_mkdata_module() 457
_os_get_blkmap() 635
_os_link() 697
_os_linkm() 699
_os_loadp() 702
_os9_allpd() 515
_os9_findpd() 624
_os9_retpd() 730
_setcrc() 927
allocate fixed-length block 515
allocate from arena 423, 444
allow access to 716
arena 423
change size of block 898
check accessibility 532
compare 453
copy 454, 455
copy external 542
create data memory module 457
create module 441
fill 456
find fixed block 624
get free block map 635
link to module 467, 469, 471, 697, 699, 702
load module 469, 471, 700, 702
into colored memory 464
make_module() 441
modcload() 464
modlink() 467
modload() 469
modloadp() 471
prevent access to block 720
request internal 358
resize block 898

resize block of 434
resize data area 707
return 271, 420
return fixed block 730
return pointer to memory block 358
re-valicate module CRC 927
search 452
set minimum allocation size 445
set reclamation bound 274
translate address 844, 846
update
CRC 927
header parity 927
memory area
clear
bzero() 150
compare
bcmp() 145
copy
bcopy() 146
memory arena 444
memory management
_cpymem() 62, 179
_freemem() 62, 272
_lcalloc() 62, 416
_lfree() 62, 420
_lmalloc() 62, 423
_lrealloc() 62, 434
_mallocmin() 63, 445
_os_alltsk() 63, 516
_os_chkmem() 532
_os_cpymem() 63, 542
_os_deltsk() 63, 559
_os_mem() 63, 707
_os_permit() 64, 716
_os_protect() 64, 720
_os_srqmem() 64, 790
_os_srtmem() 64, 794
_os9_srqmem() 64, 792
_os9_translate() 64, 846
_srqmem() 64, 968

_srtmem() 64, 970
allocate
  colored memory 64, 966
  from arena 62, 63, 423, 444
  storage for array 62, 151, 416
  task 63, 516
allow access to 64, 716
calloc() 62, 151
change size of block 898
check accessibility 532
compare 63, 453
contract data area 707
copy 63, 454
  external memory 62, 63, 179, 542
deallocate 271
  process descriptor 63, 559
determine size of unused stack area 62, 272
ebrk() 62, 198
expand data area 707
extend data segment 64, 909
fill 63, 456
free() 62, 271
freemem() 272
get external memory 62, 198
ibrk() 62, 358
list of functions 62
low-overhead 416, 420, 423, 434
malloc() 63, 444
memchr() 63, 452
memcmp() 63, 453
memcpy() 63, 454
memmove() 63, 455
memset() 63, 456
move 63, 455
prevent access to 64, 720
realloc() 64, 898
request 792
request internal memory 62, 358
resize
  block 62, 64, 434, 898
  data area 63, 707

return 271, 794
memory 62, 420
return system 64
sbrk() 64, 909
search 63, 452
set minimum allocation size 63, 445
srqcmem() 64, 966
system
request 64, 790, 968
return 64, 970
translate address 64, 846
memory map
_os9_gspump() 672
return data for 672
memory.h 20
memset() 63, 456
miscellaneous functions
list of 65
mknod() 48, 105, 459
mktemp() 48, 105, 461
mktime() 87, 462
modcload() 68, 105, 464
moddir.h 20
modes.h 21
modf() 61, 466
modlink() 68, 105, 467
modload() 69, 105, 469
modloadp() 69, 105, 471
module 645, 697
_os_altmdir() 518
_os_chain() 526
_os_chainm() 528
_os_chmdir() 534
_os_fmod() 627
_os_iodel() 682
_os_link() 697
_os_linkm() 699
_os_load() 700
_os_loadp() 702
_os_makmdir() 706
_os_modaddr() 710

_os_slink() 786
_os_slinkm() 788
_os_tlink() 840
_os_tlinkm() 842
_os_unlink() 849
_os_unload() 850
_os_vmodul() 851
_os9_vmodul() 852
_prgname() 879
_setcrc() 927
_subcall() 1029
calculate CRC 181
call subroutine module 1029
chain() 155
chainc() 155
change module directory 534
check
  CRC 851, 852
  for use of I/O module 682
  parity 851, 852
crc() 181
create 441
  module directory 706
establish alternate working directory 518
execute 155, 526, 528
find 710
  directory entry 627
get
  copy of module directory 645
  name 879
initialize static storage 678
install
  system state trap handler 840
  user subroutine library module 788
  user subroutine module 786
  user trap handler 842
link to 699, 702
load 155, 702
  from file 700
locate 710
make_module() 441

munlink() 494
munload() 495
re-validate CRC 927
set alternate working directory 518
unlink
by address 849
by name 850
from 494
unload 495
update
CRC 927
header parity 927
verify 851, 852
module directory
_get_module_dir() 313
_os_cmdperm() 537
_os_delmdir() 558
_os_fmod() 627
_os_get_mdp() 644
_os_get_moddir() 645
_os_makmdir() 706
change permissions 537
change process' current 534
create 706
delete 558
find entry 627
get
alternate pathlist 644
copy of 645
current pathlist 644
entry 313
module manipulation
_get_module_dir() 68, 313
_mkdata_module() 68, 457
_os_altmdir() 69, 518
_os_chainm() 69, 528
_os_chmdir() 69, 534
_os_cmdperm() 69, 537
_os_crc() 69, 543
_os_datmod() 69, 550
_os_delmdir() 69, 558

_os_fmod() 69, 627
_os_get_mdp() 70, 644
_os_get_moddir() 70, 645
_os_initdata() 70, 678
_os_link() 70, 697
_os_linkm() 70, 699
_os_load() 70, 700
_os_loadp() 70, 702
_os_makmdir() 70, 706
_os_modaddr() 70, 710
_os_setcrc() 70, 759
_os_slink() 70, 786
_os_slinkm() 71, 788
_os_tlink() 71, 840
_os_tlinkm() 71, 842
_os_unlink() 71, 849
_os_unload() 71, 850
_os_vmodul() 71, 851
_os9_vmodul() 71, 852
_setcrc() 71, 927
_subcall() 71, 1029
calculate module CRC 68, 181
call subroutine module 71, 1029
change
    permissions of module directory 69, 537
    process' current module directory 69, 534
crc() 68, 181
create
    data memory module 68, 457
    data module 69, 550
    module 68, 441
    new module directory 70, 706
delete existing module directory 69, 558
execute new primary module 69, 528
find
    module directory entry 69, 627
    module given pointer 70, 710
generate CRC 69, 70, 543, 759
get
    copy of module directory 70, 645
    current and alternate module directory pathlists 70, 644

module directory entry 68, 313
initialize static storage from module 70, 678
install
system state trap handling module 71, 840
user subroutine library module 71, 788
user subroutine module 786
user trap handling module 71, 842
link to memory module 68, 70, 467, 469, 471, 697, 699, 702
load
memory module 69, 70, 469, 471, 702
module into colored memory 68, 464
module(s) from file 70, 700
make_module() 68, 441
modcload() 68, 464
modlink() 68, 467
modload() 69, 469
modloadp() 69, 471
munlink() 69, 494
munload() 69, 495
re-validate module CRC 71, 927
set alternate working module directory 69, 518
unlink
from module 69, 494
module 71, 849, 850
unload module 69, 495
validate module 71, 851, 852
module.h 22, 543
move
data 711
memory 455
mq.l 11
MST 84
multibyte character
convert sequence 72, 448
convert to from wide 72, 1085
convert to from wide characters 72, 1083
determine number of bytes in 72, 446, 450
list of functions 72
mblen() 72, 446
mbstowcs() 72, 448

mbtowc() 450
wcstombs() 72, 1083
wctomb() 72, 1085
multiply float by exponent of 2 418
munlink() 69, 106, 494
munload() 69, 106, 495

N

natural logarithm 430
non-local goto 432, 933
notify driver that path has been opened 812
numeric formatting convention inquire 425

O

offsetof() 497
open
directory 501
file 256, 498
path 713
re-open file 276
open() 48, 106, 498
opendir() 48, 106, 501
os_lib.l 11, 102
os_ss_fillbuff() 806
os_ss_rfm() 818
os9def.h 23
os9exec() 78, 106, 617–620
os9fork() 78, 106, 630–633
os9forkc() 630–633
os9kexec() 78, 106, 691–696
os9time.h 23
outc() 45, 864
outl() 45, 865
output
capability string 1060
character to file 267, 883
character to standard out 883
string to file 268, 886

word to file 887
outw() 45, 866

P

parameter
_isascii() 380
_isjis_kana() 387, 389, 393
_issjis1() 401
_issjis2() 403
is it alphabetic 378
is it alphanumeric 376
is it ASCII 380
is it control code 383
is it digit 385
is it Hankaku-kana 387, 389, 393
is it hexadecimal 409
is it Kanji 401, 403
is it lower case 395
is it printable 397
is it printing character 391
is it punctuation 399
is it upper case 407
is it white space 405
isalpha() 378
isascii() 380
iscntrl() 383
isdigit() 385
isgraph() 391
islower() 395
isprint() 397
ispunct() 399
isspace() 405
isupper() 407
isxdigit() 409
parent process ID
determine 319
parse
disk file pathlist 867
path name 722
segment 882

password file
close
endpwent() 200
get entry 324
get file entry
fgetpwent() 247
get name 323, 327
get next entry for name
getnextpwnam() 314
get user identification 329
reset for searching 939
path
_gs_opt() 350
_os__ss_fillbuff() 806
_os_dup() 571
_os_prsnam() 722
_os_ss_close() 798
_os_tranpn() 843
_os9_ss_open() 812
_prsnam() 882
_ss_opt() 981
close 172, 535
copy options 350
create to new file 545, 547
dup() 196
dup2() 197
duplicate 196, 197, 571
fill buffer with data 806
get options 350
notify driver 798
of open path 812
open 713
parse name 722
segment 882
read bytes from 891, 894
read() 891
readln() 894
set options 981
translate user to system 843
valid access modes 239
write bytes to 1087, 1089

write() 1087
writeln() 1089
path descriptor
_os_getpd() 646
_os_gs_cpypd() 654
_os_gs_popt() 670
_os_ss_popt() 813
copy contents of 654
find 646
read option section 670
write option section 813
pattern
search string for 250, 252
pause() 58, 106, 869
PDT 84
perror() 67, 100, 871
pffinit() 50, 107, 872
pflinit() 50, 107, 873
pipe() 874
place
diagnostics in programs 132
formatted time in buffer 1005, 1006
position file pointer 436
pow() 61, 876
power function 876
PowerPC-specific functions
list of 90
prerr() 67, 107, 877
print
error message 201, 718, 877
to file 1073, 1075, 1077
printf() 50, 880
priority
set for process 938
setpr() 938
process
_os_aproc() 519
_os_clrsigs() 536
_os_exit() 621
_os_fork() 628
_os_forkm() 633

_os_ioexit() 683
_os_iofork() 684
_os_nproc() 712
_os_setpr() 764
_os_sigrs() 781
_os_sleep() 782
_os_suspend() 831
_os_wait() 854
_os9_aproc() 520
_os9_gspump() 672
_os9_sleep() 784
clear signal queue 536
create 628, 630, 633
end 621
insert in active process queue 519, 520
intercept() 364
kill() 413
os9fork() 630
os9forkc() 630
put to sleep 782, 784
resize current signal queue block 781
return data for memory map 672
send signal to 413
set priority 764, 938
set up
I/O 684
signal handler 364
setpr() 938
sleep 1062
start 712
suspend 831, 1079
terminate 621
I/O 683
wait for child to terminate 854, 1079
wait() 1079
process descriptor
_get_process_desc() 320
_os_alocproc() 517
_os_deltsk() 559
_os_findpd() 623
_os_get_ioproc() 643

_os_get_prtbl() 647
_os_gprdsc() 651
_os_rtnprc() 732
_os_setpr() 78
allocate 517
deallocate 559, 732
find 623
get copy 320, 651
of block table 647
initialize 517
set process priority 78
process ID
_os_id() 675
_os9_id() 677
determine 318
get 675, 677
process manipulation
_get_process_desc() 320
_get_process_table() 76, 322
_os_alocproc() 76, 517
_os_aproc() 76, 519
_os_chain() 76, 526
_os_dfork() 77, 565
_os_dforkm() 77, 569
_os_exec() 77, 611
_os_findpd() 77, 623
_os_fork() 77, 628
_os_forkm() 77, 633
_os_get_prtbl() 77, 647
_os_getstat() 77, 648
_os_getsys() 77, 649
_os_gprdsc() 77, 651
_os_id() 77, 675
_os_nproc() 78, 712
_os_rtnprc() 78, 732
_os_setpr() 764
_os_sleep() 78, 782
_os_suspend() 78, 831
_os_wait() 78, 854
_os9_allpd() 76, 515
_os9_aproc() 76, 520

_os9_dfork() 77, 567
_os9_gspump() 77, 672
_os9_id() 78, 677
_os9_sleep() 78, 784
allocate
  fixed-length block of memory 76, 515
  process descriptor 76, 517
chain() 155
chainc() 76, 155
create process 77, 78, 628, 630, 633
deallocate process descriptor 78, 732
determine parent process ID number 319
determine process ID number 76, 318
enter process in active process queue 76, 520
examine system global variable 77, 649
execute new module 76, 526
find process descriptor 77, 623
fork process under control of debugger 77, 565, 567, 569
get
  copy of process descriptor block 77
  copy of process descriptor block table 647
  file/device status 77, 648
  process descriptor copy 77, 320, 651
  process ID/user ID 77, 675, 677
  process table entry 76, 322
getpid() 76, 318
getppid() 319
insert process in active process queue 76, 519
kill() 76, 413
list of functions 76
load and execute new module 155
OS-9 system call processing 78, 617
OS-9000 system call processing 78, 691
os9exec() 78, 617
os9fork() 78, 630
os9forkc() 630
os9kexec() 78, 691
put calling process to sleep 78, 782, 784
return data for specified process' memory map 77, 672
send signal to process 76, 413
set process priority 78, 764, 938

setpr() 78, 938
sleep for specified interval 79, 1062
sleep() 78
start child process 77
start next process 78, 712
suspend
execution for specified time 78
process 78, 831
system call processing 611
tsleep() 79, 1062
wait for process termination 78, 79, 854, 1079
wait() 79, 1079
process table
_get_process_table() 322
get entry 322
process times
get 1052
process.h 24
procid.h 25
program
_exit() 232
abnormal termination 114
diagnostics 132
end 232, 233
exit() 233
termination 136
program globals 12
class 1 12
class 2 13
protection hardware
initialize 516
PST 84
put
calling process to sleep 782, 784
file descriptor sector 983
next character to file/standard out 883
putc() 50, 883
putchar() 883
putenv() 885
puts() 50, 886
putw() 50, 107, 887

Q

qsort() 80, 888
quick sort 888
quotient
    ldiv() 419
    return 193, 419

R

raise() 58, 889
rand() 61, 890
random number
    rand() 890
    return 890
    set seed for generator 965
    srand() 965
RBF caching
    _os_ss_cache() 797
    disable 797
    enable 797
rbf.h 25
read 668
    bytes from path 891, 894
    data from
        device 726
        file 269, 726
    device path options 659
    event value 588
    event without waiting 211
    file descriptor sector 663
    fread() 269
    logical unit options 668
    path descriptor option section 670
    text line 728
    word from file 339
read input
    scatter data
        readv() 896
read() 50, 107, 891
readdir() 107, 893

readln() 50, 107, 894
readv() 896
realloc() 64, 101, 898
record
  _os_ss_lock() 810
  _os_ss_ticks() 823
  _ss_lock() 979
  _ss_tiks() 990
  lock out 810, 979
  wait for release 823, 990
regcmp() 900
regerror() 901
regex() 902
regexec() 902
regexp.h 25
registers
  get value of
    get<name>() 292
  set value of 916
regs.h 26
regular expression handler
  comparison 902
  compiler 900
  error handler 901
release
  device 814, 985
  ownership of resource lock 729
remainder
  ldiv() 419
  return 193, 419
remove
  device 191, 560
    from IRQ table 686, 688
  file 903
  pending alarm request 126, 510, 742, 743
remove() 50, 100, 903
rename file 815, 904
rename() 50, 100, 904
re-open file 276
report file pointer position 289
reposition file pointer 285, 753

request
internal memory 358
system memory 792
reset
alarm clock 744
existing alarm request 511
position of directory stream 906
signal intercept context stack 779
resize
current process' signal queue block 781
data memory area 707
memory block 434, 898
resource lock
_os_acqlk() 79, 502
_os_caqlk() 79, 525
_os_crlk() 79, 549
_os_ddlk() 79, 552
_os_dellk() 79, 557
_os_rellk() 80, 729
_os_waitlk() 80, 857
acquire 79, 502
activate 80, 857
conditionally acquire ownership of 79, 525
create 79, 549
deadlock situation 79, 552
delete descriptor 79, 557
get ownership 502
list of functions 79
release ownership 80, 729
restore
device 986
head 816
interrupt level 374
retension tape drive 817
return
amount of free space on device 667
character 245
current
file size 674
interrupt level 375
location 1041

static storage pointer 335
data for memory map 672
device
name 656, 773
type 657
event information 582
exception table base address 310
file
desriptor 653
pointer to zero 905
fixed block of memory 730
free() 271
from interrupt exception 731
memory 271, 420
parent process ID 319
parts of real number 466
pointer 893
portions of floating point number 278
process ID 318
quotient and remainder 193
random integer 890
size of unused stack area 272
system
identification 833
memory 794
user ID 338
word from file 339
re-validate module CRC 927
rewind() 39, 905
rewinddir() 108, 906
rindex() 81, 108, 907
RTS line
_os_ss_dsrts() 802
_os_ss_enrts() 803
_ss_dsrts() 977
_ss_enrts() 978
disable 802, 977
enable 803, 978

S

S_IREAD 521
S_IWRITE 521
sbf.h 26
sbrk() 64, 108, 909
scanf() 50, 911
scanlist 100, 282
scanset 282
scf.h 27
SCSI device
  _os_gs_dsize() 660
  get size of 660
search
  array 147
  bit map for free area 751
  for character in string 361, 907
  memory 452
  string for pattern 250, 252
SEEK_END 285
SEEK_SET 285
seekdir() 108, 913
select() 914
semaphore
  initialize data structure for use 754
  release 757
  reserve 755
semaphore.h 27
semaphores 57
send
  data carrier
    lost signal 975
    present signal 976
  signal
    after specified seconds 118
    at Gregorian date/time 120, 504
    at Julian date/time 122, 507
    at specific time 124, 127, 506, 509, 512, 735
    on data ready 819, 988
    on DCD false 799
    on DCD true 800

to another process 758
to process 413
to program 889
system-state subroutine 740
serial connection
_os_ss_break() 796
break 796
service request table initialization 766, 768
set
alarm clock 745
alternate working module directory 518
bits in bit map 514
buffer for I/O stream 945
current file
position 287
size 987
cyclic alarm clock 738
device
path options 801
status 765
error trap handler 827, 829
event variable 213, 589, 591, 593, 597
exception table base address 929
file
attributes 795, 971
size 820
status 765, 940
global descriptor pointer 930
I/O for new process 684
IRQ level 370
memory reclamation bound 274
minimum allocation size 445
path options 981
position of next readdir 913
process
priority 764, 938
signal handler 364
relative event variable 215, 595
seed for random number generator 965
system
date 760, 762

global variable 770, 942
time 760, 762, 932
user ID 771, 944
value of registers
set<name>() 916
set up signal intercept trap 679
set user identification 931
set_excpt_base() 89, 929
set_gdtr() 89, 930
setbuf() 50, 926
setgid() 931
setime() 88, 108, 932
setjmp() 67, 933
setjmp.h 27
setlocale() 67, 935–937
setpr() 78, 108, 938
setpwent() 939
SetStat
_os_setstat() 41, 765
_os_setsys() 41, 770
_os_ss_attr() 41, 795
_os_ss_break() 41, 796
_os_ss_cache() 42, 797
_os_ss_close() 42, 798
_os_ss_dcoff() 42, 799
_os_ss_dcon() 42, 800
_os_ss_dopt() 42, 801
_os_ss_dsrts() 42, 802
_os_ss_enrts() 42, 803
_os_ss_erase() 42, 804
_os_ss_fd() 42, 805
_os_ss_fillbuf 806
_os_ss_fillbuff() 42
_os_ss_flushmap() 42, 807
_os_ss_hdlink() 42, 808
_os_ss_lock() 42, 810
_os_ss_luopt() 43, 811
_os_ss_popt() 43, 813
_os_ss_relea() 43, 814
_os_ss_rename() 43, 815
_os_ss_reset() 43, 816

_os_ss_reten() 817
_os_ss_rfm() 43, 818
_os_ss_sendsig() 43, 819
_os_ss_size() 43, 820
_os_ss_skip() 43, 821
_os_ss_skipend() 43, 822
_os_ss_ticks() 43, 823
_os_ss_wfm() 43, 824
_os_ss_wtrack() 43, 825
_os9_ss_open() 43, 812
_ss_attr() 44, 971
_ss_dcoff() 44, 975
_ss_dcon() 44, 976
_ss_dsrts() 44, 977
_ss_enrts() 44, 978
_ss_lock() 44, 979
_ss_opt() 44, 981
_ss_pfd() 44, 983
_ss_rel() 44, 985
_ss_rest() 44, 986
_ss_size() 44, 987
_ss_ssig() 44, 988
_ss_tiks() 44, 990
_ss_wtrk() 44, 992
break serial connection 41, 796
disable
    RBF caching 42, 797
    RTS line 42, 44, 802, 977
enable
    RBF caching 42, 797
    RTS line 42, 44, 803, 978
erase tape 42, 804
fill path buffer with data 42, 806
flush cached bit map information for RBF device 42, 807
list of functions 39
lock out record 42, 44, 810, 979
make hard link to existing file 42, 808
notify driver that path has been
    closed 42, 798
    opened 43, 812
put file descriptor sector 44, 983

release device 43, 44, 814, 985
rename file 43, 815
restore
device 44, 986
head to track zero 43, 816
retension pass on tape drive 817
send
data carrier
lost signal to process 44, 975
present signal to process 44, 976
signal
on data ready 43, 44, 819, 988
when DCD line goes false 42, 799
when DCD line goes true 42, 800
set
current file size 44, 987
device path options 42, 801
file
attributes 41, 44, 795, 971
size 43, 820
status 44, 940
file/device status 41, 765
path options 44, 981
system global variable 41, 770
setstat() 44, 940
skip
blocks 43, 821
tape marks 43, 818
to end of tape 43, 822
wait
for record release 44, 990
specified number of ticks for record release 43, 823
write
file descriptor sector 42, 805
logical unit options 43, 811
option section of path descriptor 43, 813
tape marks 43, 824
track 43, 44, 825, 992
setstat() 44, 108, 940
setsys.h 27
settrap.h 27

setuid() 67, 109, 944
setvbuf() 51, 945
sg_codes.h 28
sgstat.h 28
shell
    execute command 1037
    system() 1037
SIG1010 95, 954
SIG1111 95, 954
SIGABRT 93, 114, 132, 952
SIGADDR 95, 954
SIGALIGN 97, 956
SIGALRM 94, 953
SIGBNDCHK 96, 955
SIGBRKPT 96, 955
SIGCHECK 96, 97, 956
SIGCHK 95, 954
SIGCOPRCV 954
SIGDBG 96, 955
SIGDBLFLT 96, 955
SIGFMTERR 954
SIGFPE 93, 952
SIGGPROT 96, 955
SIGHUP 94, 953
SIGILL 93, 98, 952
SIGINST 96, 956
SIGINT 93, 952
SIGINVTSS 96, 955
SIGKILL 94, 953
sigmask() 58, 109, 947–949
signal
    _ev_pulse() 209
    _os_intercept() 679
    _os_send() 758
    _os_sigmask() 777
    _ss_ssig() 988
    cancel signal to send on data ready 985
    define mask 947
    event occurrence 209, 213, 215, 217, 586, 589, 591, 593, 595, 597, 599
    kill 413

mask 777
pause() 869
raise() 889
reset intercept context stack 779
send 758
on data ready 988
to program 413, 889
set up intercept trap 679
unmask 777
wait for 869
signal handler
control for process 947
defined 950
intercept() 364
set up 364
sigmask() 947
signal() 950
specify 950
signal() 58, 92, 98, 950–957
signal.h 29
signals
asynchronous 956
available 952
default handling 951
function handling 956
ignore handling 952
SIGNMI 96, 955
SIGPIPE 94, 953
SIGPRIV 95, 97, 954, 956
SIGQUIT 94, 953
SIGSEGNP 96, 955
SIGSEGV 93, 952
SIGSTACK 96, 955
SIGTERM 93, 952
SIGTRACE 95, 954
SIGTRAPV 95, 954
SIGUNIRQ 954
SIGUSR1 94, 953
SIGUSR2 94, 953
SIGWAKE 94, 953
sin() 61, 958

sine function 958
  arc 131
  hyperbolic 959
sinh() 61, 959
skip
  blocks 821
  tape marks 818
  to end of tape 822
sleep
  _os_sleep() 782
  _os9_sleep() 784
  current process 782
  for specified interval 1062
  tsleep() 1062
sleep() 78, 109
specify
  signal handling 950
sprintf() 51, 962
sqrt() 61, 964
square root function 964
srand() 61, 965
srqcmem() 64, 109, 966
sscanf() 51, 973
stack
  _freemem() 272
  _stacksiz() 994
  determine size of area 272
  freemem() 272
  get size of 994
  stacksiz() 994
stack frame
  allocate space
    alloca() 119
stacksiz() 994
standard white space characters 405
start
  next process 712
  system debugger 1036
stat() 995
static storage
  _os_initdata() 678

change current pointer 159
get_static() 335
initialize 678
return current pointer 335
status functions
_gs_devn() 344
_gs_eof() 346
_gs_gfd() 348
_gs_opt() 350
_gs_pos() 352
_gs_rdy() 353
_gs_size() 355
_os_getstat() 648
_os_getsys() 649
_os_gs_cpypd() 654
_os_gs_cstats() 655
_os_gs_devnm() 656
_os_gs_devtyp() 657
_os_gs_dopt() 659
_os_gs_dsize() 660
_os_gs_edt() 661
_os_gs_eof() 662
_os_gs_fd() 663
_os_gs_fdaddr() 664
_os_gs_fdinf() 665
_os_gs_luopt() 668
_os_gs_parity() 669
_os_gs_popt() 670
_os_gs_pos() 671
_os_gs_ready() 673
_os_gs_size() 674
_os_setstat() 765
_os_setsys() 770
_os_sgetstat() 772
_os_sgs_devnm() 773
_os_ss_attr() 795
_os_ss_break() 796
_os_ss_cache() 797
_os_ss_close() 798
_os_ss_dcoff() 799
_os_ss_dcon() 800

_os_ss_dopt() 801
_os_ss_dsrts() 802
_os_ss_enrts() 803
_os_ss_erase() 804
_os_ss_fd() 805
_os_ss_fillbuff() 806
_os_ss_flushmap() 807
_os_ss_hdlink() 808
_os_ss_lock() 810
_os_ss_luopt() 811
_os_ss_popt() 813
_os_ss_relea() 814
_os_ss_rename() 815
_os_ss_reset() 816
_os_ss_reten() 817
_os_ss_rfm() 818
_os_ss_sendsig() 819
_os_ss_size() 820
_os_ss_skip() 821
_os_ss_skipend() 822
_os_ss_ticks() 823
_os_ss_wfm() 824
_os_ss_wtrack() 825
_os9_gs_cdfd() 653
_os9_gs_free() 667
_os9_ss_open() 812
_ss_attr() 971
_ss_dcoff() 975
_ss_dcon() 976
_ss_dsrts() 977
_ss_enrts() 978
_ss_lock() 979
_ss_opt() 981
_ss_pfd() 983
_ss_rel() 985
_ss_rest() 986
_ss_size() 987
_ss_ssig() 988
_ss_tiks() 990
_ss_wtrk() 992
break serial connection 796

calculate parity of file descriptor 669
check for end of file 346
disable
  RBF caching 797
  RTS line 802, 977
enable
  RBF caching 797
  RTS line 803, 978
erase tape 804
examine system global variable 649
fill path buffer with data 806
flush cached bit map information for RBF device 807
get
  cache status information 655
  current file
    position 352, 671
    size 355
  device name 344
  file
    descriptor block address 664
    descriptor sector 348
    status 332
  file/device status 648
  I/O interface edition number 661
  path options 350
  size of SCSI device 660
  specified file descriptor sector 665
GetStat call using system path number 772
getstat() 332
lock out record 810, 979
make hard link to existing file 808
notify driver that path has been
  closed 798
  opened 812
put file descriptor sector 983
read
  device path options 659
  file descriptor sector 663
  logical unit options 668
  path descriptor option section 670
release device 814, 985

rename file 815
restore
  device 986
  head to track zreo 816
retension pass on tape drive 817
return
  amount of free space on device 667
  current file size 674
  device name 656, 773
  device type 657
  file descriptor 653
send
  data carrier
    lost signal to process 975
    present signal to process 976
  signal
    on data ready 819, 988
    when DCD line goes
      false 799
      true 800
set
  current file size 987
  device path options 801
  file
    attributes 795, 971
    size 820
    status 940
  file/device status 765
  path options 981
  system global variables 770
setstat() 940
skip
  blocks 821
  tape marks 818
  to end of tape 822
test for
  data available 353
  data ready 673
  end of file 662
wait

for record release 990
specified number of ticks for record release 823
write
file descriptor sector 805
logical unit options 811
option section of path descriptor 813
tape marks 824
track 825, 992
stdarg.h 29
stdde.h 29
stdio.h 30
stdlib.h 31
store current value of file position indicator 246
strcat() 81, 998
strchr() 81, 999
strcmp() 81, 1000
strcoll() 81, 1001
strcpy() 81, 1002
strcspn() 81, 1003
strerror() 81
strftime() 81, 88, 1005–1007
strhcpy() 81, 108, 1008
string
break into tokens 1019
catenate 998, 1010
compare 175, 1000, 1001, 1011
compute length 1015
convert to
double 1017
long 1021, 1024
copy 1002, 1012
old OS-9 strings 1008
determine length 1009
findnstr() 250
findstr() 252
get
from file 331
length 1003
index() 361
input conversion 279
locate 999

first occurrence of 1013, 1016
last occurrence of 1014
output to file 268, 886
return next option letter
getopt() 315
search for character in 361
search for first instance of pattern 252
search for first occurrence of pattern 250
strcat() 998
strchr() 999
strcmp() 1000
strcoll() 1001
strcpy() 1002
strcspn() 1003
strhcpy() 1008
strlen() 1009
strncat() 1010
strncmp() 1011
strncpy() 1012
strpbrk() 1013
strrchr() 1014
strspn() 1015
strstr() 1016
strtod() 1017
strtok() 1019
strtol() 1021
strtoul() 1024
strxfrm() 1027
transform 1027
string handling functions
break string into tokens 82, 1019
compute string length 82, 1015
copy old OS-9 strings 81, 1008
determine string length 82, 1009
findnstr() 81, 250
findstr() 81, 252
get string length 81, 1003
index() 81, 361
list of 81
locate
first occurrence of string 82, 1013, 1016

last occurrence of string 82, 1014
string 81, 999
map error message string 81
place formatted time in buffer 81, 88, 1005, 1006
rindex() 81, 907
search
for character in string 81, 361, 907
string for pattern 81, 250, 252
strcat() 81, 998
strchr() 81, 999
strcmp() 81, 1000
strcoll() 81, 1001
strcpy() 81, 1002
strcspn() 81, 1003
strerror() 81
strftime() 81, 88, 1005, 1006
strhcpy() 81, 1008
string
catenation 81, 82, 998, 1010
comparison 81, 82, 1000, 1001, 1011
copy 81, 82, 1002, 1012
to double conversion 82, 1017
to long conversion 82, 1021, 1024
strlen() 82, 1009
strncat() 82, 1010
strncmp() 82, 1011
strncpy() 82, 1012
strpbrk() 82, 1013
strrchr() 82, 1014
strspn() 82, 1015
strstr() 82, 1016
strtod() 82, 1017
strtok() 82, 1019
strtol() 82, 1021
strtoul() 82, 1024
strxfrm() 82, 1027
transform string 82, 1027
string.h 32
strings.h 32
strlen() 82, 1009
strncat() 82, 1010

strncmp() 82, 1011
strncpy() 82, 1012
strpbrk() 82, 1013
strrchr() 82, 1014
strspn() 82, 1015
strstr() 82, 1016
strtod() 82, 1017–1018
strtok() 82, 1019
strtol() 82, 1021–1023
strtoul() 82, 1024–1026
structure assignment 997
strxfrm() 82, 1027
suspend
    process 831
    task 869
svctbl.h 32
synchronous I/O multiplexing 914
sys_clib.l 11, 102
sys_mktime() 88, 108, 1031
sysglob.h 33
SYSRAM 442, 464, 709, 966
system
    _os_sysid() 833
    _os9_setime() 762
    catastrophic occurrence 715
    memory
        request 792, 968
        return 970
    return identification 833
    set date/time 762
system call processing 611, 617, 691
system debugger
    _os_sysdbg() 832
    _sysdbg() 1036
    call 832, 1036
system global variables
    _getsys() 336
    _os_getsys() 649
    _os_setsys() 770
    _setsys() 942
    change 770, 942

examine 649, 942
get 336
set 770, 942
system() 35, 1037
system_addr() 1038

T

tan() 61, 1039
tangent function 1039
arc 134, 135
hyperbolic 1040
tanh() 61, 1040
tape
_os_ss_erase() 804
_os_ss_reset() 816
_os_ss_reten() 817
_os_ss_rfm() 818
_os_ss_skip() 821
_os_ss_skipend() 822
_os_ss_wfm() 824
erase 804
restore head to track zero 816
retension 817
skip
blocks 821
marks 818
to end 822
write marks 824
task
_os_alltsk() 516
allocate 516
pause() 869
suspend 869
termination 232, 233
telldir() 109, 1041
tempnam() 1042
temporary files
create names for 1042
TERM 1043
TERMCAP 1043

Termcap
get entry 1043
tgetent() 1043
termcap.h 33
terminal
check capability reference 1045
get capability ID 1046, 1047
tgetflag() 1045
tgetnum() 1046
tgetstr() 1047
terminal manipulation functions
check terminal capability presence 83, 1045
get
cursor movement capability 83, 1049
termcap entries 83, 1043
terminal capability 83, 1047
terminal capability ID 83, 1046
list of 83
output capability string 83, 1060
tgetent() 83, 1043
tgetflag() 83, 1045
tgetnum() 83, 1046
tgetstr() 83, 1047
tgoto() 83, 1049
tputs() 83, 1060
terminal name
find
isatty() 382
terminate
_exit() 232
calling process 621
I/O for exiting process 683
program 233
task 232, 233
termio.h 33
termlib.l 11
test
_isascii() 380
_isjis_kana() 387, 389, 393
_issjis1() 401
_issjis2() 403

error indicator 242
for
alphabetic character 378
alphanumeric 376
ASCII 380
control code 383
data available 353
data ready 673
digit 385
end of file 241, 662
Hankaku-kana 387, 389, 393
hexadecimal 409
Kanji 401, 403
lowercase 395
printable character 397
printing character 391
punctuation 399
uppercase 407
white space 405
isalnum() 376
isalpha() 378
isascii() 380
iscntrl() 383
isdigit() 385
isgraph() 391
islower() 395
isprint() 397
ispunct() 399
isspace() 405
isupper() 407
isxdigit() 409
tgetent() 83, 1043
tgetflag() 83, 1045
tgetnum() 83, 1046
tgetstr() 83, 1047
tgoto() 83, 1049
time
_os9_alarm_atdate() 504
_os9_alarm_atjul() 507
broken-down structure 1031
get current 337

place formatted time in buffer 1005, 1006
send signal at Julian date/time 507
time functions
_gregorian() 87, 342
_julian() 87, 411
_os_getime() 87, 638
_os_gregorian() 88, 652
_os_julian() 88, 690
_os_setime() 88, 760
_os9_getime() 88, 640
_os9_setime() 88, 762
_sysdate() 88
asctime() 87, 129
broken-down
convert to calendar 462
clock() 87, 171
convert
broken-down to
calendar 87, 462
string format 87, 129
calendar to
Greenwich Mean Time 87, 341
local 87, 429
string format 87, 189
date/time to Gregorian value 87, 342
date/time to Julian value 87
julian date and time 411
local to Greenwich Mean Time 88, 1031
ctime() 87, 189
difftime() 87, 192
find temporal difference 87, 192
get 640, 1051
calendar time 88
current 88
Gregorian date 88, 652
Julian date 88, 690
processor time 87, 171
system date/time 87, 638
system time 87, 312
getime() 87, 312
gmtime() 87, 341

list of 87
localtime() 87, 429
mktime() 87, 462
set 88, 760, 762, 932
setime() 88, 932
sys_mktime() 88, 1031
time() 88, 1051
time zones 84
defaults 102
time() 88, 1051
time.h 33, 34
times() 1052
tmpfile() 51, 1053
tmpnam() 67, 1054
toascii() 1055
tolower() 38, 1057
toupper() 38, 1059
tputs() 83, 1060
track
_os_ss_wtrack() 825
_ss_wtrk() 992
format 825
write 825, 992
transform string 1027
translate
character 1055
memory address 846
user to system path 843
trap handler
_os_strap() 827
_os_tlink() 840
_os_tlinkm() 842
_os9_strap() 829
install system state module 840
install user module 842
set for error 827, 829
tsleep() 79, 109, 1062
TZ 84, 102

U

un.h 34
unget character 1064
ungetc() 51, 1064
unlink
  event 219, 603
  file 1066, 1067
  from module 494
  module 849, 850
unlink() 51, 109, 1066
unlinkx() 51, 109, 1067
unload module 495
unmask
  interrupts 372
  signals 777
user accounting 848
user ID
  _os_id() 675
  _os_setuid() 771
  _os9_id() 677
  determine 338
  get 675, 677
  set 771, 944
  setuid() 944
user_addr() 1069
UTC 84, 341

V

va_arg() 67, 1070
va_end() 68, 1071
va_start() 68, 1072
valid file access modes 239
validate module 852
variable parameter list
  end references to 1071
  get parameter 1070
  initialize 1072
  va_arg() 1070
  va_end() 1071

va_start() 1072
verify
attached device 143
module 851, 852
vfprintf() 51, 1073
VIDEO1 442, 464, 709, 966
VIDEO2 442, 464, 709, 966
vprintf() 51, 1075
vsprintf() 51, 1077

W

wait
for child process to terminate 854
for event to occur 221, 601, 604, 606
for process termination 1079
for record release 990
for relative event to occur 223, 607, 609
for signal 869
specified number of ticks 823
wait() 79, 109, 1079
wcstombs() 72, 1083
wctomb() 72, 1085
WET 84
write 864
bytes to path 1087, 1089
data 290, 859
file descriptor sector 805
fwrite() 290
integer to I/O address 865
line of text 861
logical unit options 811
option section of path descriptor 813
string to file 268
tape marks 824
track 825, 992
word to I/O address 866
write iovcnt buffer data 1091
write() 51, 109, 1087
writeln() 51, 109, 1089
writev() 1091

Y

YST 84

Product Discrepancy Report

To: Microware Customer Support

FAX: 515-224-1352

From:_____

Company:_____

Phone:_____

Fax:_____Email:_____

Product Name:

Description of Problem:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Host Platform_____

Target Platform_____